Fast hierarchical algorithms for generating Gaussian random fields

Pierre Blanchard, Olivier Coulaud, Eric Darve

Fast hierarchical algorithms for generating Gaussian random fields.

Pierre Blanchard, Olivier Coulaud, Eric Darve

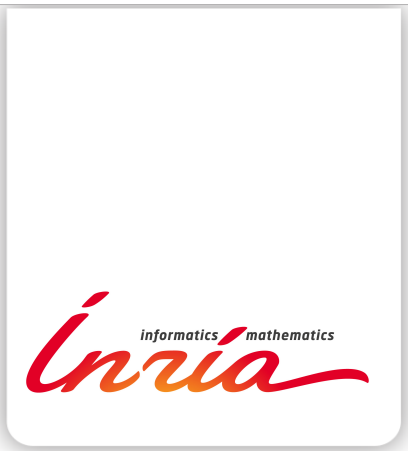
RESEARCH REPORT

N° 8811

December 2015

Project-Teams Hiepacs

ISSN 0249-6399 ISRN INRIA/RR--8811--FR+ENG



Fast hierarchical algorithms for generating Gaussian random fields.

Pierre Blanchard*, Olivier Coulaud*, Eric Darve†

Project-Teams Hiepacs

Research Report n° 8811 — December 2015 — 32 pages

Abstract: Low-rank approximation (LRA) techniques have become crucial tools in scientific computing in order to reduce the cost of storing matrices and compute usual matrix operations. Since standard techniques like the *SVD* do not scale well with the problem size N, there has been recently a growing interest for alternative methods like randomized LRAs. These methods are usually cheap, easy to implement and optimize, since they involve only very basic operations like Matrix Vector Products (MVPs) or orthogonalizations. More precisely, randomization allows for reducing the cubic cost required to perform a standard matrix factorization to the quadratic cost required to apply a few MVPs, namely $\mathcal{O}(r \times N^2)$ operations where r is the numerical rank of the matrix. First of all, we present a new efficient algorithm for performing MVPs in $\mathcal{O}(N)$ operations called the *Uniform FMM* (*ufmm*). It is based on a hierarchical (data sparse) representation of a kernel matrix combined with polynomial interpolation of the kernel on equispaced grids. The latter feature allows for *FFT*-acceleration and consequently reduce both running time and memory footprint but has implications on accuracy and stability. Then, the *ufmm* is used to speed-up the MVPs involved in the *randomized SVD*, thus reducing its cost to $\mathcal{O}(r^2 \times N)$ and exhibiting very competitive performance when the distribution of points is large and highly heterogeneous. Finally, we make use of this algorithm to efficiently generate spatially correlated multivariate Gaussian random variables.

Key-words: $\mathcal{H}^2$-methods, *FMM*, *FFT*, *randomized SVD*, covariance kernel matrices, multivariate Gaussian random variables.

* Inria, Hiepacs Project, 350 cours de la Libération, 33400 Talence, France. Email: Surname.Name@Inria.fr
† Mechanical Engineering Department, Institute for Computational and Mathematical Engineering, Stanford University, 452 Escondido Mall, Building 520, Room 125, Stanford, CA 94305, USA. Email: darve@stanford.edu

RESEARCH CENTRE
BORDEAUX – SUD-OUEST

200 avenue de la Vieille Tour
33405 Talence Cedex

Algorithmes hiérarchiques rapides pour la génération de champs aléatoires Gaussiens.

Résumé : Les approximations de rang faible (LRA) sont devenus des outils fondamentaux en calcul scientifique en vue de réduire les coûts liés au stockage et aux opérations matricielles. Le coût des méthodes standards comme la SVD croît très rapidement avec la taille du problème N, c'est pourquoi des méthodes alternatives comme les approches aléatoires (i.e. basées sur la projection ou la sélection de colonnes/l'échantillonnage aléatoire) se popularisent. Ces méthodes sont en général peu coûteuses et facile à implémenter et optimiser, car elles ne mettent en oeuvre que des opérations matricielles simples comme des produits ou des orthogonalisations. Plus précisemment, les LRA aléatoires permettent de réduire le coût cubique en N des méthodes standards de factorisation au coût quadratique nécessaire à la réalisation de quelques produits matrices vecteurs, *i.e.*, $\mathcal{O}(r \times N^2)$ opérations où r est le rang numérique de la matrice. Dans un premier temps, nous présentons un algorithme efficace pour réaliser des MVPs en $\mathcal{O}(N)$ opérations, que nous appelons *Uniform FMM* (*ufmm*). Il est basé sur la combinaison d'une représentation hiérachique d'une matrice noyau et l'interpolation polynomiale du noyau associé sur une grille régulière (uniforme). Cette dernière propriété permet une accélération par *FFT* réduisant ainsi le temps de calcul et la consommation mémoire mais a des répercussions sur la précision et la stabilité de l'algorithme. Ensuite, la *ufmm* est utilisée pour accélerer les MVPs intervenants dans la SVD aléatoire (*i.e.* SVD par projection aléatoire) diminuant son coût asymptotique à $\mathcal{O}(r^2 \times N)$. La méthode est particulièrement compétitive pour des distributions de points hétérogènes. Enfin, nous utilisons cet algorithme pour générer des champs de variables Gaussiens de manière efficace.

Mots-clés : Méthodes $\mathcal{H}^2$, *FMM*, *FFT*, *randomized SVD*, matrices de covariance, champs aléatoires Gaussiens.

1 Introduction

Generating realizations of Gaussian Random Fields (GRFs) is a key issue in numerous scientific research fields such as cosmology [42, 7, 14], geostatistics [28, 30], hydrogeology [43, 6], brownian dynamics [5],... A GRF is a multivariate Gaussian random variable, *i.e.*, a vector $\mathbf{Y} \in \mathbb{R}^N$ of correlated Gaussian random variables each associated with the points of a grid denoted $\{x_i\}_{i=1...N}$. In the present work we will only consider spatial grids, *i.e.*, $x_i \in \mathbb{R}^3$. In order to accurately represent the statistic of the problem, GRFs usually need to be generated in large ensembles, *e.g.*, we need $\mathcal{O}(10^5)$ realizations to reach errors between 1 and 10% in term of experimental mean $\mathbb{E}(\mathbf{Y})$ and covariance $\mathbb{E}(\mathbf{Y}\mathbf{Y}^\mathbf{T})$. Therefore, efficiently generating many realizations of a GRF given a large spatial grid, *i.e.*, $N = \mathcal{O}(10^6)$, can rapidly become problematic even on modern computers.

Correlation kernels We use the terminology $\mathbf{Y} \sim \mu(\mathbf{0}, \mathbf{C})$ to define a GRF $\mathbf{Y}$ with mean $\mathbf{0}$ and covariance $\mathbf{C} \in \mathbb{R}^{N\times N}$. The covariance can be prescribed as a kernel matrix, *i.e.*,

$$\mathbf{C} = \{k(r_{ij})\}_{i,j=1...N} \tag{1}$$

, where $r_{ij} = |x_i - x_j|$ and k is a correlation kernel such as

$$k_{1/2}(\mathbf{r}) = e^{-|\mathbf{r}|\ell} = e^{-r/\ell} \qquad \text{(Exponential decay)}$$
$$k_\infty(\mathbf{r}) = e^{-|\mathbf{r}|_\ell^2/2} = e^{-r^2/2\ell^2} \qquad \text{(Gaussian decay)}$$

These functions are the extreme cases of Matérn functions k_ν, where the length scale ℓ characterizes the decreasing speed of the correlation. Covariance matrices are symmetric positive definite (*spd*) by definition of correlation kernels. Therefore $\mathbf{C}$ admits a square root $\mathbf{A}$, *i.e.*,

$$\mathbf{C} = \mathbf{A}\mathbf{A}^\mathbf{T} \tag{2}$$

A standard approach Realizations of a correlated GRF $\mathbf{Y} \sim \mu(\mathbf{0}, \mathbf{C})$ can be obtained by applying $\mathbf{A}$ to a white noise $\mathbf{X} \sim \mu(\mathbf{0}, \mathbf{I_N})$, *i.e.*, a Gaussian random field $\mathbf{X}$ verifying $\mathbb{E}(\mathbf{X}) = \mathbf{0}$ and $\mathbb{E}(\mathbf{X}\mathbf{X}^T) = \mathbf{I_N}$. There exists numerous approaches for generating GRFs, that usually differ by the way the square root $\mathbf{A}$ is precomputed. Approaches based on standard matrix decompositions such as the Cholesky Decomposition (*CD*) [16] are the most popular ones, since they provide exact square roots and rely on well understood and robust algorithms that apply to any *spd* matrix $\mathbf{C}$. They can also provide fast (*i.e.*, $\mathcal{O}(N)$) approximate methods to generate GRFs, if the decomposition is truncated at a prescribed numerical rank, *i.e.*, $\mathbf{A} \in \mathbb{R}^{N\times r}$. However standard factorization algorithms involve $\mathcal{O}(N^3)$ operations and thus become computationally prohibitiv for large N, *i.e.*, N over a few thousands. Alternative methods are often considered such as the sequential simulation method [27], the moving average methods [40], the turning bands method [28, 36], continuous [45] or discrete [25, 43] spectral methods , combinations of spectral methods with the turning bands [35] or with moving average [30], improved matrix decomposition [15]... They usually provide approximate square roots but most importantly they do not always extend (well or at all) to 3D grids.

FFT approach A popular alternative based on matrix decomposition, often referred to as the *FFT* approach (or *Circulant Embedding*), was introduced in 1993 by Dietrich and Newsam [18] and simultaneously developed by Wood and Chan [51]. It provides an exact method for computing the product $\mathbf{Y} = \mathbf{A}\mathbf{X}$, that has a $\mathcal{O}(N)$ cost in both running time and storage. The square root $\mathbf{A}$ is assembled in Fourier domain in $\mathcal{O}(N \log N)$ operations. However it presents some significant numerical limitations especially in 3D as explained in [18], not to mention that it only applies to equispaced grids.

1.1 Randomized LRA

Randomized algorithms for the low-rank approximation (LRA) of matrices, *a.k.a.*, randomized LRA algorithms, usually provide powerful alternatives to standard matrix factorizations for matrices of relatively low-rank. Recently, they have gained popularity in the numerical linear algebra community because they are easy to implement, highly parallelizable and most of all they achieve very competitive performance within reasonable accuracy. Moreover, these algorithms often involve very basic matrix computations thus leaving significant room for improvement. They have drawn much attention in scientific fields such as geostatistics [29, 17], machine learning [19, 50], genetics [34],... For further references on their applications to data analysis please refer to Mahoney [33]. Randomized algorithms divide in 2 classes: random sampling- and random projection-based algorithms.

Random projection In the present paper we focus on the randomized *SVD*, a very popular *random projection*-based LRA algorithm introduced and further enhanced in a series of papers [38, 32, 53]. A comprehensive review of the method, extensions to other factorizations such as *CD* or Interpolative Decompositon (*ID*), as well as fast variants can be found in Halko et al. [26]. Let us simply recall that the randomized *SVD* of a *symmetric* matrix $\mathbf{C} \in \mathbb{R}^{N\times N}$ is a 2-stage process:

- A *randomized range approximation*: Form an approximate basis $\mathbf{Q} \in \mathbb{R}^{N\times r}$ for the range of $\mathbf{C}$ using a Gaussian random projection, *i.e.*, application of $\mathbf{C}$ to a N-by-r Gaussian random matrix $\mathbf{\Omega}$
$$\mathbf{Y} = \mathbf{C\Omega} \tag{3}$$
and a subsequent orthogonalization, *e.g.*, a QR decomposition (*QRD*). Thus, we get a low-rank representation of $\mathbf{C}$ in the form
$$\mathbf{C_r} = \mathbf{QQ^TCQQ^T} \tag{4}$$
- The second stage consists in factorizing $\mathbf{C_r}$ in *SVD* form: $\mathbf{C_r} = \mathbf{U\Sigma U^T}$. We start by assembling the small r-by-r matrix
$$\mathbf{B} = \mathbf{Q^TCQ} \tag{5}$$
then perform an *SVD*, *i.e.*, $\mathbf{B} = \mathbf{U_B\Sigma_B U_B^T}$.

Finally, an approximate *SVD* of $\mathbf{C}$ is obtained by forming $\mathbf{U} = \mathbf{QU_B}$ and $\mathbf{\Sigma} = \mathbf{\Sigma_B}$. Alternatively, if $\mathbf{C}$ is symmetric positive semi-definite (*spsd*), a square root is obtained as
$$\mathbf{A} = \mathbf{QB^{1/2}} \tag{6}$$
with $\mathbf{B^{1/2}} = \mathbf{U_B\Sigma^{1/2}}$. Since the overall cost of the randomized *SVD* is dominated by the cost of the matrix multiplication required in each stage, the algorithm has a $\mathcal{O}(N^2 \times r)$ asymptotic complexity where r represents the prescribed numerical rank. In the context of GRF simulations, Dehdari and Deutsch [17] used the randomized *SVD* in order to accelerate the precomputation of A in low-rank form and thus efficiently generate realizations of GRF at a $\mathcal{O}(N \times r)$ cost in running time. This approach stillrequire $\mathbf{C}$ to be fully assembled

1.2 Contributions: a randomized *SVD* powered by a new $\mathcal{H}^2$-method

In the present paper we describe a new approach for precomputing low-rank square roots of covariance kernel matrices. It combines the randomized *SVD* with a new efficient hierarchical matrix-multiplication algorithm.

An *FFT*-accelerated $\mathcal{H}^2$-method: *ufmm* Our first contribution, introduced in the preliminary section 2, consists in an highly efficient *FFT*-accelerated $\mathcal{H}^2$-method for computing Matrix-to-Vector Products (MVPs) with kernel matrices in linear time called the *Uniform FMM* or *ufmm*[1]. This polynomial interpolation-based $\mathcal{H}^2$-method, derived from the *black-box FMM* (*bbfmm*) by Fong and Darve [21], relies on equispaced interpolation grids which dramatically decreases the amount of data to be stored and allows for *FFT*-acceleration. The accuracy of the associated interpolation scheme is expected to be slightly worse than the *bbfmm*; however, in most cases of interest including the one studied here, this loss of accuracy is offset by performance of the *FFT*. The algorithm is implemented within the ScalFMM package, an open-source generic parallel *FMM* library available online at http://scalfmm-public.gforge.inria.fr/doc/[2], along with several other schemes including the original *FMM* [23, 24] and the *bbfmm* schemes [21, 39].

A randomized *SVD* powered by ufmm Our second contribution, described in Section 3, consists in using the *ufmm* to perform the matrix multiplications involved in the randomized *SVD*. This provides an algorithm for approximating the square root of a covariance kernel matrix in $\mathcal{O}(r^2 \times N)$ operations instead of the original $\mathcal{O}(r \times N^2)$. The resulting approach is *matrix-free*, thus reducing the complexity to $\mathcal{O}(r \times N)$ in terms of memory requirements and allowing very large grids to be handled. Furthermore, exploiting the hierarchical structure of the matrix allows highly heterogeneous grids to be computed faster. The accuracy of the square root algorithm remains unchanged if the extra error introduced by the *ufmm* is carefuly monitored. Since our approach is fairly generic, it extends to other randomized matrix factorization (*e.g.*, *CD*, *ID*) and more generally to any dense random projection based LRA algorithm, as long as the input matrix C is suited for $\mathcal{H}^2$-methods. The algorithm is implemented within the FMR package, an open-source library for computing fast randomized LRAs available online at https://gforge.inria.fr/projects/fmr. The package provides both standard and fast techniques for generating GRF based on matrix decomposition, namely *FFT*, *CD*, *SVD*, *RandSVD* and *Nystrom* method.

Numerical benchmarks The accuracy and the sequential performance of the algorithm were tested on numerical benchmarks. Results are presented in Section 4 in order to confirm the theoretical linear complexity of our algorithm and compare the performance with the original randomized *SVD*. Finally, we verify that the method accurately generates GRFs on various artificial test cases.

1.3 Related works

Various aspects of our algorithms relate to other existing works in the fields of deterministic and randomized LRA. These relations are summarized and discussed in this section.

1.3.1 *FFT* conversion of an $\mathcal{H}^2$-method

An *FFT* conversion of the original Fast Multipole Algorithm (FMA) was originally proposed in a technical report by Greengard et al. [24], and later implemented and improved by Elliott et al. [20]. For an efficient parallel implementation of this approach please refer to Pan et

[1] In order to stay consistent with the terminology used in our implementations and past publications, we decided to keep the name *Uniform FMM* although *Lagrange FMM* might seem more consistent with related methods such as *Chebyshev FMM*. Moreover, the terminology *Uniform FMM* emphasizes the regularity of the equispaced interpolation grid that allows for *FFT*-acceleration.

[2] A complete documentation is provided online along with links to associated publications, see [8, 3, 2] for further information on the package.

al. [41]. The general idea of the original *FFT* conversion proposed in these papers consists in rewritting the `M2L` operations in the form of a 2D linear convolution, then converting it to a 2D circular convolution (using zero-padding) in order to finally perform the `M2L` operations in Fourier domain at a lower cost using *FFT*. This approach differs from our method since it is based on expanding the kernel $1/r$ in spherical harmonics while our method relies on *kernel-independent* polynomial interpolations. On the other hand, both variants require zero-padding for the conversion to circular convolution. The *FFT* conversion of the FMA algorithm additionally suffers from *smearing effects*, *i.e.*, some entries are artificially not zeroed out due to the conversion to circular convolution form. While our method suffers from the Runge phenomenon due to the interpolation on an equispaced grid, the original variant suffers from various sources of instability. The first occurs when a coefficient with strong variations in magnitude is warped in order to rewrite the expansion as a linear convolution. The second instability results from a scaling problem that amplifies during the *FFT* operations. Stabilization techniques were proposed in [24] and further discussed and improved in [20]. In the present paper, we also suggest methods in order to defeat the Runge phenomenon that may occur in our method.

1.3.2 Fast randomized LRA

The acceleration of randomized LRA algorithms based on *random projection* usually relies on the ability to apply fast Matrix-to-Matrix Products (MMPs), hence most research works in this area focus on exploiting the structure and the sparsity of the input matrix C and the random matrix Ω.

Random matrix Random projection algorithms exploiting the nature of the random matrix are presented by Woodruff [52] in a comprehensive survey within the general framework of matrix sketching. While Sarlós [44] first mentioned the idea of using structured random (sketching) matrices such as *Fast Johnson-Lindenstrauss Transforms* (*FJLT*), Ailon et al. [4] described the *FJLT* algorithm in full details. The idea is based upon Achlioptas's work [1] that aimed at developing sparse sketching for more efficient random projection. Woolfe et al. [53] subsequently designed an *FJLT* algorithm based on the Discrete Fourier Transform (*DFT*), namely the Subsampled Randomized Fourier Transform (*SRFT*). The ability to compute such transform in quasi-linear time allowed for reducing the computational cost of the random projection from $\mathcal{O}(N^2 \times r)$ to $\mathcal{O}(N^2 \times \log r)$. Such method can be used to compute LRAs of matrices, in fact in Liberty et al. [32] the *SRFT* is used to speedup the randomized *ID* introduced in Martinsson et al. [38]. However, it is not clear yet if there exists a fixed accuracy variant for fast transform based projection schemes. Various efficient algorithms for performing dense random projection are studied in Liberty's thesis [31] such as the general *FJLT*, the *SRFT*, the *SRHT* (based on the Discrete Walsh-Hadamard Transform) as well as the Mailman algorithm for matrix multiplication. Recently, Tropp [47] provided an improved analysis for the *SRHT*.

Input matrix To our knowledge, only very few papers address fast matrix multiplication for *Gaussian random projection*-based algorithm applied to dense structured input matrices. In the context of low-rank approximations of matrices, Martinsson [37] describes a way to build HSS matrices using a *FMM*-powered *Gaussian random projection* algorithm. The *FMM* is used there in order to accelerate the projection stage, *i.e.*, the product between the input matrix and the Gaussian random matrix (Eq. 3). Then, the factorization in HSS form is handled by ID. In the present paper, we use *FMM*-accelerated MMPs in order to speedup the projection (Eq. 3) but also to build the matrix B (Eq. 5) efficiently.

Random sampling Random sampling techniques are particularly well suited for LRA applications requiring relatively low accuracy. The most widely used random sampling algorithm for *spd* matrices is the Nyström method, see [19, 22, 54]. Applications of the Nyström method to the LRA of covariance kernel matrices can be found in Wang et al. [48]. An improved variant of the original Nyström method introduced in Si et al. [46] exploiting the block structure of kernel matrices is known as the *MEKA*. It relies on a preclustering of the data and a block-wise LRA. Very recently, Wang et al. [49] provided an algorithm based on a similar idea and called the *Block Basis Factorization* (*BBF*), that does not have stability issues and exhibits smaller standard deviation than the *MEKA*. Such method can provide low-rank matrix square roots for covariance kernel matrices in $\mathcal{O}(N)$ operations with a low dependence on the ambiant dimension. Since it is taylored for the high dimensions involved in machine learning applications, the *BBF* should perform worse in 3D applications than our analytical interpolation schemes. Finally, [48] introduces its own $\mathcal{O}(N)$ randomized *SVD*, namely a subsampled randomized *SVD*.

2 Preliminary: An *FFT*-accelerated $\mathcal{H}^2$-method, the *ufmm*

In this section we present a new variant of the *black-box FMM* [21] (or *bbfmm*) where the interpolation is done on a uniform (*i.e.*, equispaced) grid and the scheme is accelerated by *FFT*, we call this new method the *uniform FMM* or *ufmm*. We also propose adjustements to the original algorithm in order to increase its efficiency for globally smooth kernels. The algorithms of the *bbfmm* 2 and the *ufmm* 3 are given in Appendix 6.

2.1 A new interpolation based $\mathcal{H}^2$-method

Let us consider a system of N particles interacting with each other. We want to efficiently compute the contributions of all N *source* particles on each *target* particle $i = 1, \dots, N$, *i.e.*, in less than $\mathcal{O}(N^2)$ operations. If we denote $\mathbf{x}_i$ the position of particle i, then its potential $f(\mathbf{x}_i) = f_i$ due to the densities $w(\mathbf{x}_j) = w_j$ can be expressed as follows

$$f(\mathbf{x}_i) = \sum_{j=1}^{N} k(\mathbf{x}_i, \mathbf{x}_j) w(\mathbf{x}_j) \tag{7}$$

where $k(\cdot, \cdot)$ represents the kernel of interactions.

Hierarchical clustering The *bbfmm* is a $\mathcal{H}^2$-method for computing sums like (7) based on a hierarchical partitionning of the domain using a cluster tree structure, namely an octree. The root cluster $\mathcal{C}^{(0)}$ of the tree is the smallest cube enclosing all particles. At the level L of the octree we subdivide all *parent* cells $\mathcal{C}^{(L)}$ in 8 cubes of equal size to get all *child* cells $\mathcal{C}^{(L+1)}$ at level $L+1$. This recursive partition is stopped at the *leaf* level $\bar{L}$ once a suitable criterion is reached (*e.g.*, minimum or average number of particles per leaf cell, minimum leaf cell size).

Interaction list The method relies on an octree in order to partition the domain in clusters and then identifies admissible cluster pairs, *i.e.*, clusters of particles whose interactions can be approximated using a low-rank approach. More precisely, two cells of width ω distant from D form an admissible cell pair if they satisfy the admissibility criterion $D \geq \gamma\omega$, where γ can be prescribed ($\gamma = 2$ in the original *bbfmm*). Let $\mathcal{C}^{(L)}$ denote an arbitrary cell at level L. Cells that form admissible pairs with $\mathcal{C}^{(L)}$ are also said to be in farfield interactions with $\mathcal{C}^{(L)}$, while the other are in nearfield interactions with $\mathcal{C}^{(L)}$. The neighbor list $\mathcal{N}(\mathcal{C}^{(L)})$ is defined as the

set of cells in nearfield interactions with $\mathcal{C}^{(L)}$. The interaction list $\mathcal{I}(\mathcal{C}_{\mathbf{x}}^{(L)})$ is defined as the set of non-empty cells $\mathcal{C}_{\mathbf{y}}^{(L)}$ in farfield interactions with $\mathcal{C}_{\mathbf{x}}^{(L)}$ such that the parent of $\mathcal{C}_{\mathbf{x}}^{(L)}$ and $\mathcal{C}_{\mathbf{y}}^{(L)}$ are in nearfield interactions. The latter condition ensures that all contributions are only computed once during the hierarchical summation scheme. Figure 1 illustrates how interaction and neighbor lists are built on the last two levels of a 1D tree structure.

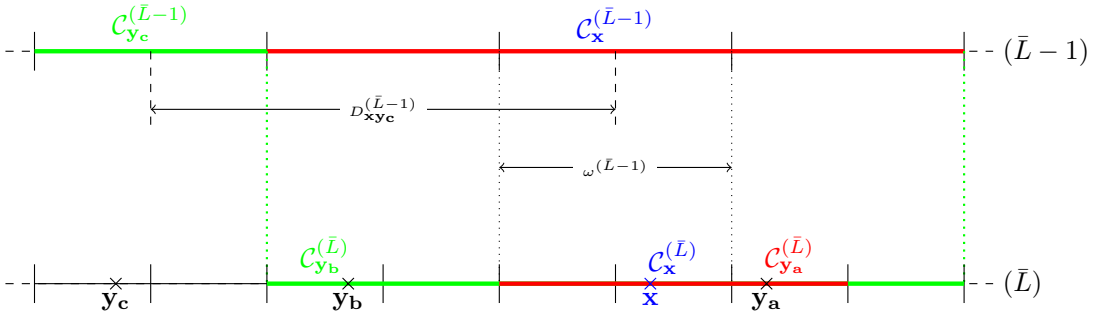


Figure 1: Portion of a 1D tree at level $\bar{L}$ and $\bar{L}-1$. The interaction lists $\mathcal{I}(\mathcal{C}_{\mathbf{x}}^{(L)})$ (green) and neighbor lists $\mathcal{N}(\mathcal{C}_{\mathbf{x}}^{(L)})$ (red) of a target particle $\mathbf{x}$ is represented for $\gamma = 2$. The contribution of the source particle $\mathbf{y_a}$ is computed analytically as part of the nearfied while contributions of $\mathbf{y_b}$ and $\mathbf{y_c}$ are approximated as part of the farfield at level $\bar{L}$ and $\bar{L}-1$ respectively.

A general interpolation scheme The LRA technique involved in the *bbfmm* is a general interpolation scheme based on Chebyshev polynomials. While most LRA methods discussed in this paper are purely algebraic, this approach requires the kernel $k(\cdot,\cdot)$ to be known explicitly. However, as opposed to methods relying on *kernel-specific* expansions like the spherical harmonics for $k(r) = 1/r$, methods based on general interpolation schemes are *kernel-independent*. For any source $\mathbf{y}$ and target $\mathbf{x}$ lying in admissible clusters, the kernel $k(\mathbf{x},\mathbf{y})$ can be approximated using the following polynomial interpolation formula

$$k(\mathbf{x},\mathbf{y}) \approx \sum_{|\boldsymbol{\alpha}|\leq p} S_{\boldsymbol{\alpha}}^{p}(\mathbf{x}) \sum_{|\boldsymbol{\beta}|\leq p} K_{\boldsymbol{\alpha}\boldsymbol{\beta}} S_{\boldsymbol{\beta}}^{p}(\mathbf{y}) \tag{8}$$

where $S_{\boldsymbol{\alpha}}^{p}$ is referred to as a 3D polynomial interpolator (of order p) and $K_{\boldsymbol{\alpha}\boldsymbol{\beta}}$ denotes the evaluation of $k(\cdot,\cdot)$ at interpolation points $\bar{\mathbf{x}}_{\boldsymbol{\alpha}}$ for targets and $\bar{\mathbf{y}}_{\boldsymbol{\beta}}$ for sources. The multi-index notations is defined as follows

$$\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \alpha_3), \text{ with } \alpha_i = 0, \ldots, p \text{ and } |\boldsymbol{\alpha}| = \max_{i\leq 3}(\alpha_i)$$

Error bounds for (8) are provided in [21]. The sum (7) can thus be approximated by

$$f(\mathbf{x}_i) \approx \sum_{|\boldsymbol{\alpha}|\leq p} S_{\boldsymbol{\alpha}}^{p}(\mathbf{x}_i) \sum_{|\boldsymbol{\beta}|\leq p} K_{\boldsymbol{\alpha}\boldsymbol{\beta}} \sum_{j=1}^{N} S_{\boldsymbol{\beta}}^{p}(\mathbf{x}_j) w(\mathbf{x}_j)$$

with $S_{\boldsymbol{\alpha}}^{p} = S_{\alpha_1}^{p} \times S_{\alpha_2}^{p} \times S_{\alpha_3}^{p}$ where S_n^p is a 1D polynomial interpolator.

Computational cost Algorithm 2 presents the original *bbfmm* summation scheme as introduced by Fong and Darve [21]. The cost of both storing and applying the $(p+1)^3 \times (p+1)^3$ matrices $\bar{\mathbf{K}} = \{K_{\boldsymbol{\alpha}\boldsymbol{\beta}}\}_{|\alpha|,|\beta|\leq p}$ (*a.k.a.*, `M2L` operators) to the expansions is $\mathcal{O}(p^6)$. Moreover these matrices must be applied a potentially large number of times (max. 189 interactions per cell per level), which usually makes the `M2L` step the most computationally expensive step of any Fast Multipole scheme. In order to accelerate the *bbfmm* Messner et al. [39] described several optimizations based on compressing the `M2L` operators and exploiting their symmetries.

2.2 Acceleration by *Fast Fourier Transform* (*FFT*)

In this paper, we present a new optimized hierarchical summation scheme based on a Lagrange polynomial interpolation in place of Chebyshev. This scheme is described in Algo. 3 and will now be referred to as the *Uniform FMM* or *ufmm*. As described in the present section, Lagrange interpolation combined with the *Fast Fourier Transform* (*FFT*) allows for dramatically decreasing the memory footprint and the computational cost of the `M2L` operators, namely from $\mathcal{O}(p^6)$ to $\mathcal{O}(p^3 \log p)$. In 1D, the Lagrange interpolators S_n^p take the following form

$$S_n^p(x) = L_n^p(x) = \prod_{\substack{m=0 \\ m\neq n}}^{p} \frac{x - \bar{x}_m}{\bar{x}_n - \bar{x}_m}, \; \forall x \in [-1, 1]$$

for $n = 0, \ldots, p$ where $\bar{x}_m = -1 + 2m/p$ for $m = 0, \ldots, p$. In our implementations we actually used a slightly different form that reads as

$$S_n^p(x) = \frac{(-1)^{p-n}}{2^p n!(p-n)!} \prod_{\substack{m=0 \\ m\neq n}}^{p} \left(p(x+1) - 2m\right), \; \forall x \in [-1, 1]$$

in order to reduce the effects of round-off errors.

Addressing Runge phenomenon Interpolation schemes based on equispaced grids usually become unstable for high orders of interpolation p (Runge phenomenon), however as described in a review by Boyd et al. [11] efficient regularization methods have already been introduced. They are for instance based on optimization (for Tikhonov regularization see [9]), series expansions (for Gegenbauer regularization see [10]), subsampling (for overdetermined least-squares and Mock-Chebyshev interpolation see [13]) or even multi-domain approaches (see [12]). These methods defeat Runge phenomenon while preserving subgeometric convergence. In our approach divergence is not likely to occur since we use interpolation on multiple subdomains and the value of p remains relatively low ($p < 20$) in almost all cases of interest. In particular, machine accuracy has been reached with our algorithm for a wide range of non-oscillatory kernels, *e.g.*, $1/r$, $1/r^2$, correlations (Matérn functions, spherical model) and various isotropic elastostatic Green's functions.

Structure of the `M2L` operators On the other hand, performing the interpolation on an equispaced (*uniform*) grid allows for an easy conversion of the algorithm to Fourier domain and thus leading to a faster scheme. Indeed when $k(\cdot, \cdot)$ is evaluated on a uniform 1D grid then $\bar{\mathbf{K}}$ is a Toeplitz matrix, *i.e.*, each diagonal contains constant values. This is a consequence of the fact that the entries of $\bar{\mathbf{K}}$ only depend on the distance between the corresponding particles, *i.e.*, $k(x_i, x_j) = k(x_i - x_j)$. In the case of 2D grids the resulting matrix is block Toeplitz, *i.e.*, the matrix is composed of constant blocks over its diagonals while each block is itself Toeplitz. In 3D

we include an extra level of blocking meaning that the constant diagonal blocks are now block Toeplitz.

Circulant embedding Dietrich [18] and Wood [51] almost simultaneously proposed a scheme in which a Toeplitz matrices is embedded in a larger circulant matrix and then applied as a convolution in Fourier space. We briefly recall this method in the case of a 1-dimensional Toeplitz `M2L` operator $\bar{\mathbf{K}}$ of order p. For the sake of clarity let us consider the case where $\bar{\mathbf{K}}$ is symmetric, then $\bar{\mathbf{K}}$ is fully defined by its first row

$$\mathbf{R} = (\rho_0, \dots, \rho_p)^t \in \mathbb{R}^{p+1}, \text{ with } \rho_i = k(\bar{x}_0, \bar{x}_i) \text{ for } i = 0, \dots, p$$

This row can be embedded into the first row $\tilde{\mathbf{R}}$ of a (symmetric) circulant matrix $\bar{\mathbf{E}} \in \mathbb{R}^{\tilde{p}\times\tilde{p}}$ with $\tilde{p} = p + 1 + p - 1 = 2p$ such that

$$\tilde{\mathbf{R}} = (\rho_0, \dots, \rho_p, \rho_{p-1}, \dots, \rho_1)^t = (\mathbf{R}, \rho_{p-1}, \dots, \rho_1)^t \in \mathbb{R}^{\tilde{p}}$$

By construction $\bar{\mathbf{K}}$ is embedded in the upper left corner of $\bar{\mathbf{E}}$, *i.e.*, the application of both matrices is equivalent if the last $p-1$ columns and rows of $\bar{\mathbf{E}}$ are masked. In the non-symmetric case the embedding is slightly larger, namely $\tilde{p} = p+1+p = 2p+1$, *i.e.*, the number of components that fully define the non-symmetric matrix $\bar{\mathbf{K}}$.

Conversion to Fourier domain The discrete convolution theorem implies that the set of eigenvectors of any circulant matrix $\bar{\mathbf{E}}$ forms the Discrete Fourier Transform (DFT) operator, *i.e.*, $\mathbf{F} = \left\{\tilde{p}^{-1/2} e^{-2i\pi mn/\tilde{p}}\right\}_{m,n=0,\dots,\tilde{p}}$, while the vector $\mathbf{\Lambda}$ containing the eigenvalues is known to be the DFT of the first column of $\bar{\mathbf{E}}$, *i.e.*, $\mathbf{\Lambda} = \mathbf{F}\tilde{\mathbf{R}}$. Let us consider a multipole expansion $\mathbf{M} \in \mathbb{R}^{p+1}$ and a resulting local expansion $\mathbf{L} = \bar{\mathbf{K}}\mathbf{M} \in \mathbb{R}^{p+1}$. If $\tilde{\mathbf{M}}$ denotes the expansion obtained after padding $\mathbf{M}$ with $p-1$ zeros, then for $i = 0, \dots, p$ we have

$$(\mathbf{L})_i = \left(\bar{\mathbf{E}}\tilde{\mathbf{M}}\right)_i = \left(\mathbf{F}^*\mathbf{diag}(\mathbf{\Lambda})\mathbf{F}\tilde{\mathbf{M}}\right)_i = \left(\mathbf{F}^*\left[\mathbf{\Lambda} \odot \mathbf{F}\tilde{\mathbf{M}}\right]\right)_i = \left(\mathbf{F}^*\left[\mathbf{F}\tilde{\mathbf{R}} \odot \mathbf{F}\tilde{\mathbf{M}}\right]\right)_i$$

Hence, the matrix vector product $\mathbf{L} = \bar{\mathbf{K}}\mathbf{M}$ can be performed in Fourier space in the form of an entrywise product $\mathbf{F}\tilde{\mathbf{R}} \odot \mathbf{F}\tilde{\mathbf{M}}$. Transfers back and forth between Fourier and physical domains are done by *FFT* resulting in an asymptotic overall cost of $\mathcal{O}(p^d \log p)$ for precomputation and $\mathcal{O}(p^d)$ for application, where d denotes the ambiant dimension.

Numerical complexity The theoretical complexities of the *bbfmm* and *ufmm* algorithms are given in Table 1. They are not only given in term of computational cost but also in term of memory footprint for the `M2L` (always precomputed) and the `P2P` (precomputed if matrix to matrix operations are considered). In the *ufmm* the precomputation of the `M2L` operators requires $\mathcal{O}(p^3 \log p)$ operations (*i.e.*, the cost of an *FFT*) while their application requires only $\mathcal{O}(p^3)$ operations (*i.e.*, the cost of an entrywise product). All algorithms scale linearly in N but they optimize the interpolation steps differently in particular during the most expensive step of the method, namely the `M2L` step. In fact, the cost of storing and applying `M2L` operators scales like $\mathcal{O}(p^3)$ in the *ufmm* and $\mathcal{O}(p^6)$ in the *bbfmm*, therefore we expect significant differences in memory requirements and computational times.

2.3 A block low-rank algorithm for smooth kernels: *smooth-ufmm*

The *Fast Multipole Method* (*FMM*) introduced by Greengard et al. [23] was originally developed for kernels with a strong singularity at the origin, *e.g.*, $k(x, y) = 1/|x-y|$. This property leads to

	bbfmm	*ufmm*	*smooth-ufmm*
`P2P` (CPU/memory)	$n_0 \times N$	$n_0 \times N$	0
`P2M/L2P` (CPU)	$p^3 \times N$	$p^3 \times N + p^3 \log(p) \times N/n_0$	$p^3 \times N + p^3 \log(p) \times N/n_0$
`M2M/L2L` (CPU)	p^4	$p^4 + p^3 \log(p)$	$p^4 + p^3 \log(p)$
build `M2L` (CPU)	p^6	$p^3 \log(p)$	$p^3 \log(p)$
apply `M2L` (CPU)	p^6	p^3	p^3
`M2L` (memory)	p^6	p^3	p^3

Table 1: Asymptotic complexities of the algorithms. The complexities of the `M2L` and `M2M/L2L` are given per level and per non empty cell, they should be multiplied by the number of non-empty cells ($\mathcal{O}(2^{3L})$) and then summed over levels. For the `P2P` and `P2M/L2P` the complexities are given globally. The constant $n_0 = N/2^{3\bar{L}}$ denotes the average number of particles per leaf.

the design of a multilevel scheme that allowed for approximating a larger part of the field without ever reaching the singularity. Most correlation kernels do not present any strong singularity (*e.g.*, Matérn family) and the Gaussian k_∞ is even perfectly smooth. More precisely, the larger is ν the smoother is k_ν near the origin.

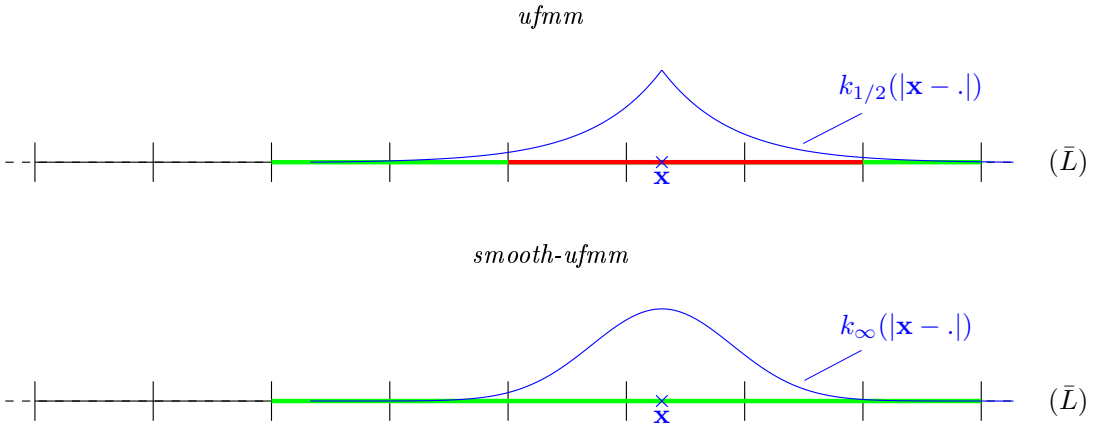


Figure 2: Schematic view of both *ufmm* variants on a 1D tree. The target particle $\mathbf{x}$ (blue) lies in the leaf cell $\mathcal{C}^{\bar{L}}_{\mathbf{x}}$. The interaction list $\mathcal{I}(\mathcal{C}^L_{\mathbf{x}})$ at a given level L is represented in green, while the nearfield $\mathcal{N}(\mathcal{C}^L_{\mathbf{x}})$ is represented in red.

smooth-ufmm Provided the kernel is *sufficiently* smooth near the origin we want to consider a summation scheme where the nearfield interactions are approximated as well and thus no direct interaction is ever computed. An algorithm of this nature can be derived from the *ufmm* algorithm by changing the admissibility criterion $\gamma^{(\bar{L})}$ at the leaf level from 2 to 0. While in the *ufmm* well-separation of leaves ($\gamma^{(\bar{L})} = 2$) is imposed, in the new variant coinciding leaves form admissible pairs as well ($\gamma^{(\bar{L})} = 0$). In fact, since the kernel is *sufficiently* smooth at the origin then the diagonal subblocks of the kernel matrix associated with coincident and adjacent cell pairs are *relatively* low-rank. The interaction list of a given leaf now includes the direct neighbors (and the leaf itself), which means that all interactions will be transferred by means of `M2L` operations and thus no interaction needs to be computed at the `P2P` step. This variant is denoted *smooth-ufmm* and can be seen as a block low-rank approximation technique. Figure 2

illustrates the difference between both variants in terms of interaction lists. In the *smooth* variant the interaction list at the leaf level $\mathcal{I}(\mathcal{C}_x^{\bar{L}})$ includes all leaf cells whose parent is a direct neighbor of the target parent cell $\mathcal{C}_x^{\bar{L}-1}$ and consequently $\mathcal{N}(\mathcal{C}_x^{\bar{L}}) = \varnothing$, whereas in the standard variant the direct neighbors of the target leaf cell form the nearfield.

Optimal setup Since the `P2P` and the `M2L` do not compete anymore in the *smooth-ufmm*, these steps do not have to be balanced. Consequently, the concept of level is not relevant anymore and the actual tuning parameter becomes the width of the leaf cells. More precisely, having bypassed the `P2P` step, we only need to minimize the cost of the `M2L` step by using the fewest number of clusters, *i.e.*, the largest leaf cells. However, as our algorithm remains a multi-level scheme, the depth for the octree is still used to control the width of the leaves. The optimal setup for the algorithm is the lowest tree depth leading to a similar accuracy as the original *ufmm* scheme.

Computational cost Due to the extension of the interaction list, the maximum number of `M2L` operators to store and apply at the leaf level increases from 189 to $189 + 27 = 216$ in 3D. However, as shown in the numerical benchmarks presented in Section 4.1, for a given accuracy the number of level is usually lower than the level required for the standard *ufmm*. Moreover, since no direct computation is involved, the computational time of the *smooth-ufmm* is expected to be lower than for the *ufmm*. These algorithms are used here for computing MMPs; so, it is crucial to precompute and store the `P2P` operators. Therefore, the *smooth-ufmm* requires a significantly lower amount of memory than the *ufmm* (see memory requirements in Table 1).

3 A fast randomized algorithm for generating GRFs

In this section we present our new algorithm for efficiently generating GRFs on large spatial grids given a correlation kernel. We start by describing the original randomized *SVD* approach. Then, we explain how to setup some parameters in order to get optimal performance in the desired range of accuracy. Finally, we present our new fast algorithm, for approximating a square root $\mathbf{A}$ of a covariance matrix $\mathbf{C}$, namely the $\mathcal{H}^2$-powered randomized *SVD* (Algorithm 1).

3.1 Precomputation of $C^{1/2}$ via randomized *SVD*

The original randomized *SVD* As explained in section 1, a *random projection*-based LRA such as the randomized *SVD* [26] is a 2-stage process: a randomized range approximation based on dense Gaussian random projection followed by a factorization in standard form, namely an *SVD*. The *Randomized Subspace Iterations* (*RSI*, Algorithm 4.4 in [26]) can be used to approximate the range given a prescribed rank r. However, since in our case the rank is not known in advance and accuracy has to be monitored, we prefer to use an adaptive-rank or fixed accuracy variant such as the *Adaptive Randomized Range Finder* (*ARRF*, Algorithm 4.1 in [26]). Indeed the *ARRF* returns a *near-optimal* range approximation given a prescribed accuracy ε. Due to their random nature these algorithms have a non-zero chance to fail, however error bounds such as 9 generally hold with high probability, *e.g.*, at least $1 - N \times 10^{-r}$ for the *RSI* algorithm.

Tune-up parameters and accuracy In order to improve the accuracy of the range approximation the projection can be performed on a slightly larger subspace, namely using a N-by-$(r+s)$ random matrix $\mathbf{\Omega}$ where s is called the oversampling parameter, and finally keeping the first r columns of $\mathbf{Q}$. Furthermore, the original algorithm does not apply well to matrices that have slow decreasing (or flat) spectrum because it fails to identify the most important singular values. A

common alternative is to stretch this spectrum by using q subspace iterations, *i.e.*, by projecting $(\mathbf{CC}^*)^q\mathbf{C}$ instead of $\mathbf{C}$. The resulting $\mathbf{C_r}$ (Eq. 4) approximates $\mathbf{C}$ within well-established and controlled error bounds, that can be expressed in Frobenius norm as

$$\mathbb{E}(\|\mathbf{C} - \mathbf{C_r}\|_F) \leq f_F(r, s, q)\|\mathbf{C} - \mathbf{C_r}\|_F^{opt.} \tag{9}$$

The deterministic lower bound $\|\mathbf{C} - \mathbf{C_r}\|_F^{opt.}$, *a.k.a.*, the baseline in [26], reads as

$$\|\mathbf{C} - \mathbf{C_r}\|_F^{opt.} = \left(\sum_{i=r+1}^{N} \sigma_i^2(\mathbf{C})\right)^{1/2} \tag{10}$$

where f_F is a polynomial function of the rank r that ensures tighter error bounds in average Frobenius norm as s and q grow, see [26] for a comprehensive discussion on error bounds.

3.2 Reaching *near-optimal* accuracy

Oversampling Figure 3 illustrates the benefit of using oversampling on the Gaussian kernel k_∞. No power iteration is required, since the Gaussian correlation has a fast decreasing spectrum. Hence, we set $q = 0$ and analyze the effect of the oversampling. First of all, the results show that the fixed rank approach works well for matrices of relatively low rank, *e.g.* for $\ell \geq 0.50$. For instance, on the unit sphere with $N = 2000$, $r = 100$ (*i.e.*, $r/N \approx 5\%$) and $s = 5$, if $\ell \geq 0.50$ then the relative error in Frobenius norm is below 2%. Moreover, as shown on Figure 3, increasing the oversampling significantly improves the accuracy and even provide *near-optimal* accuracy on this particular test case. In fact, for $\ell \geq 0.5$, the error for $s = 50$ almost coincides with the theoretical lower bound $\|\mathbf{C} - \mathbf{C_r}\|_F^{opt.}$.

Subspace iterations On the other hand, Figure 4 illustrates the benefit of using power iterations on the exponential kernel, *i.e.*, $k_{1/2}$, that exhibits a relatively flat spectrum. In particular, it shows that with an oversampling of $s = 50$ the accuracy is still far from the optimal but *near-optimal* accuracy can be reached with a few ($q = 2$) power iterations. Since in this case the matrix is not low-rank, it only produces a raw approximation of $\mathbf{C}$ in the desired range of compression, *i.e.*, below 10%. Hence, we will not consider this kernel in the numerical benchmarks presented in Section 4.

3.3 An $\mathcal{H}^2$-powered randomized *SVD*

Algorithm & complexity The overall cost of the original randomized *SVD* is quadratic in N as it is dominated by the matrix-to-matrix products (MMPs) involved in each stage, namely $\mathbf{C\Omega}$ for the random projection (Equation 3) and $\mathbf{CQ}$ for the factorization (Equation 5). Since $\mathbf{C}$ is given as a kernel matrix it can be applied to another matrix at a linear cost in N without ever assembling the full matrix using a $\mathcal{H}^2$-method like the *ufmm* (or *bbfmm*). The resulting approach remains a dense random projection as $\mathbf{\Omega}$ stays dense, but it benefits from a data sparse representation of $\mathbf{C}$. Algorithm 1 presents our $\mathcal{H}^2$-powered randomized *SVD* and indicates the asymptotic complexity of each stage. The resulting algorithm has an overall $\mathcal{O}(r^2 \times N)$ computational cost.

Managing accuracy $\mathcal{H}^2$-methods provide approximate matrix multiplications, therefore they introduce extra errors in each stage of the algorithm. The accuracy of the $\mathcal{H}^2$-method needs to be carefuly monitored in order to avoid perturbing the approximation of the range and the final

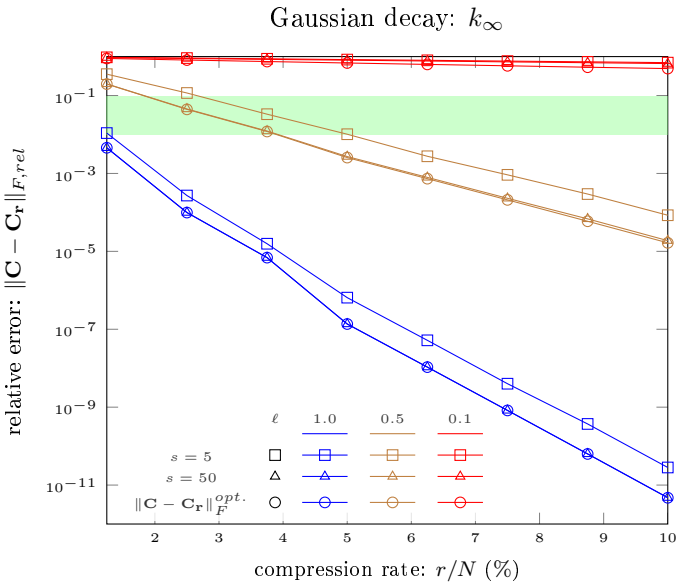


Figure 3: Accuracy of the fixed rank *RandSVD w.r.t.* the compression rate for a Gaussian correlation ($N = 2000$ and $r = 20 \dots 200$). We analyze the effects of the oversampling s alone ($q = 0$). Observations: The Gaussian kernel is smooth, therefore its spectrum decreases fast and $\mathbf{C}$ is relatively low-rank. Consequently, the *RandSVD* performs well even without power iterations and with low oversampling ($s = 5$). An oversampling of $s = 50$ leads to a near optimal error. Note that $\mathbf{C}$ becomes high-rank for small ℓ, *e.g.*, $\ell = 0.1$.

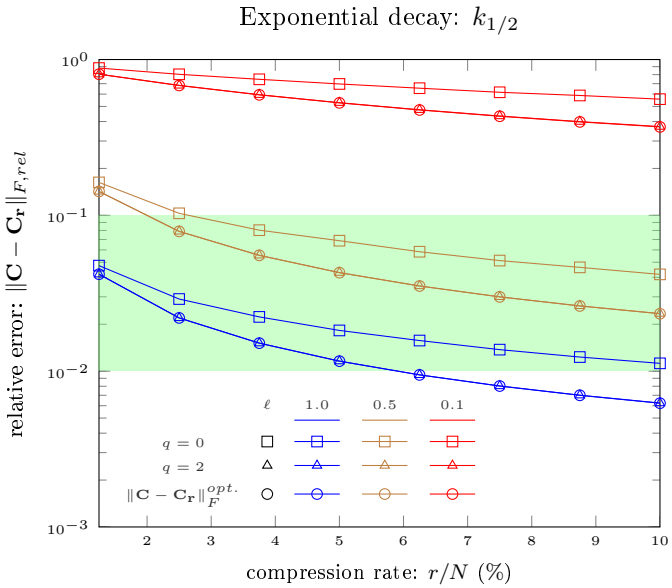


Figure 4: Accuracy of the fixed rank *RandSVD w.r.t.* the compression rate for an exponential correlation ($N = 2000$ and $r = 20 \dots 200$). We analyze the effects of q power iterations using a fixed oversampling of $s = 50$. Observations: The exponential correlation $k_{1/2}$ is not smooth, therefore its spectrum is relatively flat. A few power iterations lead to a near optimal error but the rank remains relatively high. $\mathbf{C}$ becomes high-rank for small ℓ, *e.g.*, $\ell = 0.1$.

Algorithm 1: $\mathcal{H}^2$-powered Randomized SVD

Input: Correlation kernel $k(.,.)$, grid size N, Positions $\mathbf{x}$, rank r or accuracy ε, number of power iterations q, oversampling parameter s, interpolation order p, octree depth h

Output: $[\mathbf{U}, \mathbf{\Sigma}]$ approximate SVD of $\mathbf{C} = \{k(x_i, x_j)\}_{i,j<N}$

// Stage I: Approximate the range of $\mathbf{C}$

if *accuracy ε is prescribed* **then**

  $[\mathbf{Q}, r] = ARRF(k, x, N, p, h, q, s, \varepsilon)$ using $\mathcal{H}^2$-MVPs. $\mathcal{O}(r^2 \times N)$

if *rank r is prescribed* **then**

  $[\mathbf{Q}, \varepsilon] = RSI(k, x, N, p, h, q, s, r)$ using $\mathcal{H}^2$-MVPs. (idem)

// Stage II: Decompose $\mathbf{C_r} = \mathbf{Q}(\mathbf{Q^T C Q})\mathbf{Q^T} \approx \mathbf{C}$ as $\mathbf{U\Sigma U^T}$

Build $\mathbf{B} = \mathbf{Q^T C Q} \in \mathbb{R}^{r \times r}$ using $\mathcal{H}^2$-MVPs. $\mathcal{O}(r^2 \times N)$

Perform SVD of $\mathbf{B} = \mathbf{U_B \Sigma_B U_B^T}$ $\mathcal{O}(r^3)$

Form $\mathbf{U} = \mathbf{Q U_B}$ and $\mathbf{\Sigma} = \mathbf{\Sigma_B}$ $\mathcal{O}(N \times r^2)$

// Compute square-root $\mathbf{A} = \mathbf{C_r^{1/2}} \approx \mathbf{C^{1/2}}$ as $\mathbf{U\Sigma^{1/2}}$

$\mathbf{B^{1/2}} = \mathbf{U_B \Sigma_B^{1/2}}$ $\mathcal{O}(N \times r^2)$

$\mathbf{A} = \mathbf{Q B^{1/2}}$ $\mathcal{O}(N \times r^2)$

accuracy of the low-rank representation. The optimal order of the interpolation is determined so that it does not significantly affect the output rank of the $ARRF$ or the accuracy of the RSI. More precisely, we found that enforcing one order of magnitude between the error on a fast MVP $\varepsilon_{ufmm} = \|f_{ufmm} - Kw\|_{2,rel}$ and the prescribed accuracy ε, *i.e.*,

$$\varepsilon = 10\varepsilon_{ufmm} \tag{11}$$

preserves the accuracy of the original method while still providing significantly better performance.

Efficient implementation We use the ScalFMM library to perform the fast matrix multiplications involved in the algorithm. Since ScalFMM's kernels were originally designed to perform fast MVPs, we made use of the multi right-hand-side feature to apply MMPs one block after another. The amount of column vectors applied at once to the matrix $\mathbf{C}$ can be set to any value n_{rhs}. In the present work, we used $n_{rhs} = 10$ in order to limit the memory required to store the multipole and local expansions. As a result, we need about 1GBytes in order to store the expansions at the leaf level for a tree height of $h = 5$ and $p = 5$ (in double precison arithmetic). Furthermore, the nearfield interactions (*a.k.a.*, $P2P$ operators) are assembled and stored in memory in order to be applied faster using level 3 BLAS routines. This can affect the storage significantly for large N, *i.e.*, above 10^6 particles, therefore the trade-off between memory consumption and running time has to be adressed with great care. In our implementation, if the memory required by the P2P exceeds a certain limit then we recommend using a larger tree height h. On the other hand, if the *smooth-ufmm* is used then we only need to store the $M2L$ operators and the expansions.

4 Numerical benchmarks

In this section, we compare the performance of our fast algorithm compared to the *RandSVD* in order to confirm the theoretical complexities and measure the actual speedup of the method. We first do a comparative analysis between various interpolation based $\mathcal{H}^2$-methods. Then, we confirm the complexity of the *ufmm*-powered *RandSVD*. Finally, we generate large ensemble of GRFs in order to verify the accuracy of the approach and discuss the running times.

Test cases We only consider the Gaussian correlation kernel, *i.e.*, $k = k_\infty$, since its smoothness allows for using the very efficient *smooth-ufmm* and since it provides relatively low-rank covariance matrices. The particles are either distributed on the surface of the unit sphere (heterogeneous distribution, hence many cells of the octree are empty) or inside a $2 \times 2 \times 2$ cube (homogeneous distribution, hence octree is densely populated). Both distributions share a common root bounding box of width $w^0 = 2$. See Appendix 7 for more heterogeneous distributions of particles. All experiments were conducted in double precision arithmetic (ScalFMM and FMR also implements single precision).

4.1 Comparison of $\mathcal{H}^2$-methods for fast matrix multiplication

We first compare the relative accuracy and numerical performance of the *bbfmm* and the *ufmm* on MVPs for $k(x, y) = 1/|x - y|$. Then, we compare the *ufmm* with the *smooth-ufmm* on MMPs for a Gaussian correlation kernel $k = k_\infty$ with $\ell = 0.5$.

bbfmm vs ufmm The comparative cost of the *bbfmm*, with or without compression of the `M2L` operators (see [39]), and the *ufmm* is represented Fig. 5. The *ufmm* outperforms the unoptimized *bbfmm* in terms of both computational time and memory requirements. Let us recall that the *symmetric bbfmm* is taylored for symmetric kernels allowing for a massive reduction of the interaction list and that it involves individual compression of the `M2L` operators, while the *compressed bbfmm* only involves a global compression of the `M2L` operators. As expected the *ufmm*

is slightly less accurate than the *bbfmm* (due to the near minimax property of the Chebyshev interpolation) but still performs better for a given precision. As shown on the graphs the *ufmm* and the *symmetric bbfmm* have similar performance though the *ufmm* applies to any kernel. More precisely the *ufmm* is faster in terms of computational times but requires a little more memory than the *symmetric bbfmm*.

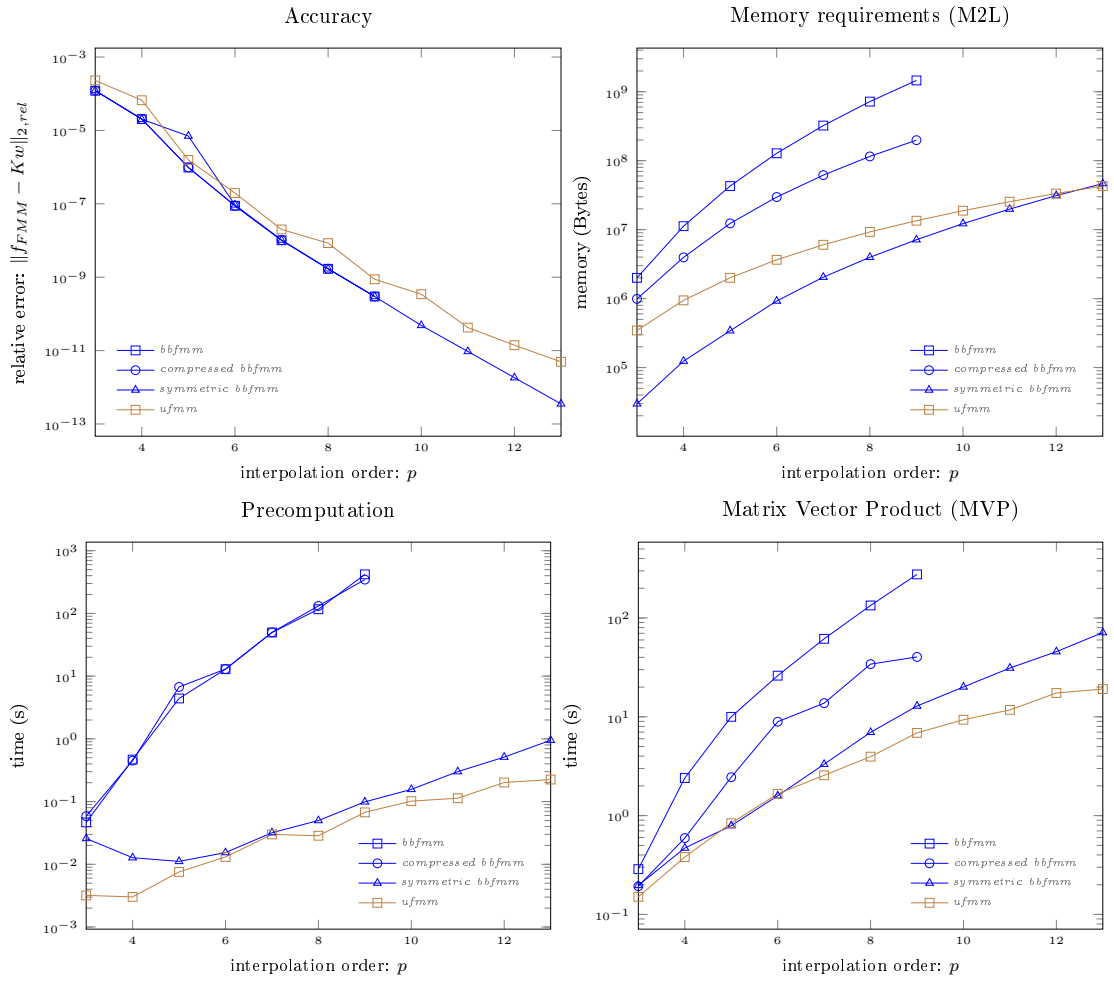


Figure 5: Accuracy and performance of the *ufmm* and variants of the *bbfmm* with respect to the interpolation order p. We used $k(x, y) = 1/|x - y|$ and 20,000 particles randomly distributed in the $2 \times 2 \times 2$ cube. Observations: For a given accuracy, the *ufmm* performs better than generic variants of *bbfmm* in terms of both computational time and memory footprint. Its performance are close to the *bbfmm* optimized for symmetric kernels. Machine: desktop computer - Intel Core i7-3520M CPU @ 2.90GHz x 4 with 8GB ram.

ufmm vs smooth-ufmm In the graphs Fig. 6 we compare the relative performance of the *ufmm* and the *smooth-ufmm* for a given accuracy of about 10^{-3}. The *smooth-ufmm* unsurpris-

ingly has better performance than the *ufmm*, since the cost of approximating the nearfield must become cheaper as N increases compare to the *ufmm* with an increasing tree depth.

global fft The *smooth-ufmm* with $h = 1$ boils down to an *FFT*-accelerated Lagrange interpolation on the bounding box. It is referred to as the *global fft* as it generalizes the idea of *FFT* on an arbitrary grid. Therefore, for a given bounding box, the performance of the *global fft* are independent of the shape of the distribution. As shown on Fig. 6, even in this range of accuracy, the global approach is almost always slower than the hierarchical variants that furthermore benefit from the heterogeneity of the distribution. Figures shown in Appendix 7 confirm these observations for a fixed N and a varying accuracy.

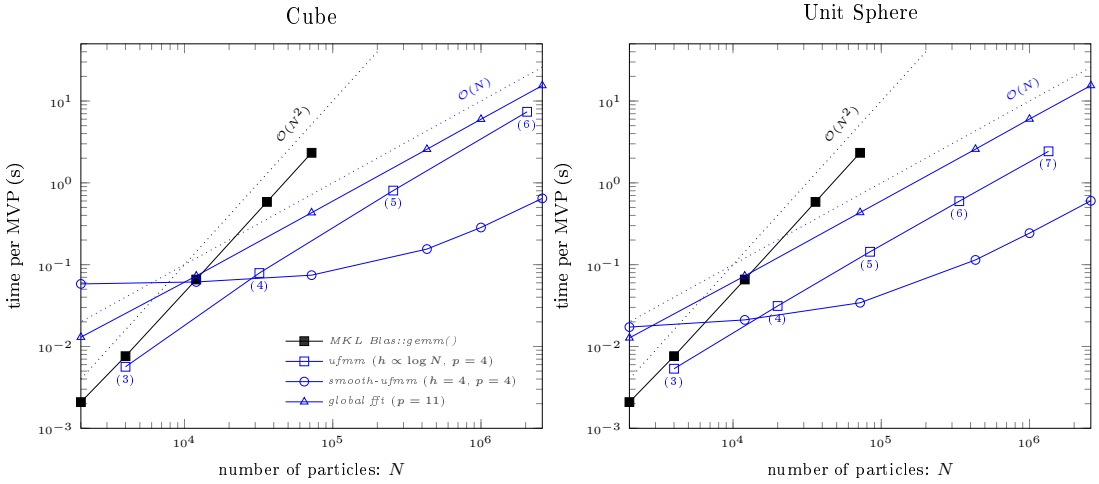


Figure 6: Time per MVP for a total of 10 MVPs using either *ufmm* (with pre-assembled `P2P` operators) or *smooth-ufmm*. We choose p such that the relative L2 error is below 10^{-3}. Particles are either randomly distributed in the $2 \times 2 \times 2$ cube (left) or on the unit sphere (right). The tree depth h of the *ufmm* (written below blue circles) ensures a relatively constant average number of particles per leaf: $n_0 = 74$ (left) and $n_0 = 62$ (right). The correlation is Gaussian with $\ell = 0.5$. Machine: plafrim/riri - Deca-core Intel Xeon E7-4870 @ 2.40GHz with 1TB ram and 30MB L3 Cache.

4.2 performance of the $\mathcal{H}^2$-powered *RandSVD*

The accuracy and the $\mathcal{O}(N)$ complexity of the $\mathcal{H}^2$-powered *RandSVD* are now illustrated for a Gaussian correlation with $\ell = 0.5$ and particles distributed on the unit sphere.

Optimal setup As shown by Figure 3 the covariance matrix can be represented by a matrix of rank $r \approx 70$ with a precision of about $\varepsilon = 10^{-2}$ in Frobenius norm using a conventional *RandSVD*. In this case, our experiments showed that the smallest interpolation order that does not affect the randomized algorithm is $p = 4$ for the *ufmm*, *i.e.*, $\varepsilon_{ufmm} < 10^{-3}$. If the desired accuracy equals $\varepsilon = 10^{-3}$ then $r \approx 100$ and $p = 5$ was found optimal, *i.e.*, $\varepsilon_{ufmm} < 10^{-4}$. This confirms the optimal setup rule proposed in Equation 11.

Complexity Figure 7 shows the time required to build an approximate square root of $\mathbf{C}$ within an accuracy of 10^{-2} in Frobenius norm. It confirms the theoretical asymptotic complexities of all approaches. As mentioned in [26], we observe that the running times of the fixed rank and fixed accuracy variants are very close to each other. Moreover, the $\mathcal{H}^2$ variants (*ufmm* and *smooth-ufmm*) are faster than the original *RandSVD* for $N \geq 10^4$. Not only do these variants allow computation to be performed with N up to a few millions but they also provide significant speedup, *e.g.*, they would be about $10\times$ faster for $N = 10^5$ if only the dense computation was affordable. This acceleration is even more pronounced for smooth kernels and heterogenous grids as explained in Section 4.1.

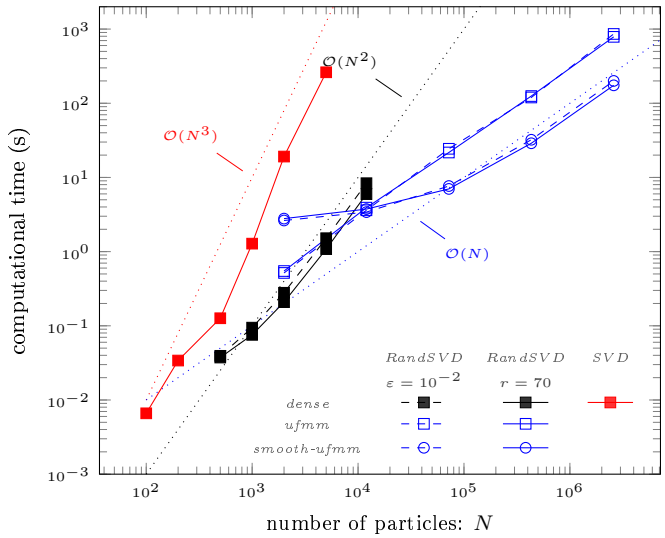


Figure 7: Time for computing a randomized *SVD* using either the fixed rank or fixed accuracy algorithm with $q = 0$ and $s = 10$. MMPs are computed either in a dense way (filled black squares) or by means of the *ufmm* (empty squares) or *smooth-ufmm* (empty circles) with $p = 4$, particles are randomly distributed on the unit sphere and the correlation is Gaussian with $\ell = 0.5$. Dense MMPs, *SVD* and *QRD* are performed using sequential MKL Blas routines. Observations: As expected, both fixed rank and fixed accuracy variants exhibit similar performance. On the other hand, the graphs confirm the theoretical asymptotic costs (linear in blue, quadratic in black and cubic in red). Machine: plafrim/riri - Deca-core Intel Xeon E7-4870 @ 2.40GHz with 1TB ram and 30MB L3 Cache.

4.3 Realizations of Gaussian Random Fields

Gaussian Random Fields were simulated on particles distributed on the unit sphere using Gaussian correlations with varying length scales ℓ. The approximation is done by mean of a fixed precision ($\varepsilon = 10^{-2}$) *RandSVD* with *ufmm*-accelerated MVPs. Realizations are displayed on Figure 8 for various length scales.

Accuracy The sample covariance matrix $\tilde{\mathbf{C}}$, computed from the realizations as

$$\tilde{\mathbf{C}} = \frac{1}{n_{real}} \sum_{i=1}^{n_{real}} (\mathbf{Y}_i - \mathbb{E}(\mathbf{Y}_i))(\mathbf{Y}_i - \mathbb{E}(\mathbf{Y}_i))^t \tag{12}$$

provides a good approximation of the experimental covariance. Therefore we analyze the accuracy of the method using the error between $\tilde{\mathbf{C}}$ and the actual covariance matrix $\mathbf{C}$. In order to limit the computational cost required by verifications, the error is computed on a subset of the matrices. The accuracy of the method *w.r.t.* the number of realizations n_{real} is presented in Table 2. They show that *ufmm*- and *smooth-ufmm*-accelerated *RandSVD* provide square roots that generate correlated random fields with similar accuracies and a convergence rate that is close to the theoretical 1/2.

Running times As shown by figure 6, for a Gaussian correlation with $\ell = 0.50$, building an approximate square root with $N = 72k$, $r = 70$ takes about 20 seconds with the *ufmm*, 8 seconds with the *smooth-ufmm* while it would take about 5 minutes with dense MMPs. For $\ell = 0.25$, the rank r and the computational times are about 3 times larger.

Gaussian k_∞ with $\ell = 0.25$ Gaussian k_∞ with $\ell = 0.50$

Figure 8: Realization of a Gaussian random field on $72k$ points distributed on the unit sphere using a Gaussian correlation. The approximate square root was obtained by mean of a *smooth-ufmm*-accelerated *RandSVD* ($\varepsilon = 10^{-2}$).

5 Conclusion

We presented a new optimized $\mathcal{H}^2$-method for computing fast MMPs, namely an *FFT*-accelerated variant of the *bbfmm*. The method is very efficient compared to other optimized *bbfmm* in both computational time and memory footprint. The algorithm has been optimized for smooth matrix kernels allowing for further improvement of the performance. This hierarchical MMP method is implemented within a randomized *SVD* resulting in a significant acceleration of the algorithm for low-rank kernel matrices. We used this hierarchical randomized *SVD* to compute approximate low-rank square roots of covariance matrices and generate correlated Gaussian Random Fields at a reasonable cost compared to standard methods. The resulting approach is faster than most existing alternatives and has many interesting features (*e.g.*, *kernel-independent*, *matrix-free*, easy to parallelize, ...) but obviously requires the input matrix to be relatively low-rank. Finally, we want to emphasize that the benefits of using a hierarchical randomized *SVD* to approximate square roots of covariance matrices were demonstrated on a very basic and well-studied problem, however the method is inherently generic and thus extends to a broad range of applications.

$\ell = 0.25$

ufmm

n_{real}	$\|\tilde{C} - C\|_{2,rel}$	roc_2	$\|\tilde{C} - C\|_{\infty,rel}$	roc_∞
$1 \cdot 10^3$	$2.48 \cdot 10^{-1}$		$1.47 \cdot 10^{-1}$	
$1 \cdot 10^4$	$7.88 \cdot 10^{-2}$	0.5	$4.53 \cdot 10^{-2}$	0.51
$1 \cdot 10^5$	$2.63 \cdot 10^{-2}$	0.48	$1.57 \cdot 10^{-2}$	0.46
$1 \cdot 10^6$	$1.02 \cdot 10^{-2}$	0.41	$8.53 \cdot 10^{-3}$	0.27

smooth-ufmm

n_{real}	$\|\tilde{C} - C\|_{2,rel}$	roc_2	$\|\tilde{C} - C\|_{\infty,rel}$	roc_∞
$1 \cdot 10^3$	$2.48 \cdot 10^{-1}$		$1.57 \cdot 10^{-1}$	
$1 \cdot 10^4$	$7.88 \cdot 10^{-2}$	0.5	$5.68 \cdot 10^{-2}$	0.44
$1 \cdot 10^5$	$2.67 \cdot 10^{-2}$	0.47	$1.84 \cdot 10^{-2}$	0.49
$1 \cdot 10^6$	$1.09 \cdot 10^{-2}$	0.39	$1.09 \cdot 10^{-2}$	0.23

$\ell = 0.50$

ufmm

n_{real}	$\|\tilde{C} - C\|_{2,rel}$	roc_2	$\|\tilde{C} - C\|_{\infty,rel}$	roc_∞
$1 \cdot 10^3$	$1.31 \cdot 10^{-1}$		$1.12 \cdot 10^{-1}$	
$1 \cdot 10^4$	$3.75 \cdot 10^{-2}$	0.54	$4.56 \cdot 10^{-2}$	0.39
$1 \cdot 10^5$	$1.41 \cdot 10^{-2}$	0.43	$1.76 \cdot 10^{-2}$	0.41
$1 \cdot 10^6$	$4.72 \cdot 10^{-3}$	0.47	$5.95 \cdot 10^{-3}$	0.47

smooth-ufmm

n_{real}	$\|\tilde{C} - C\|_{2,rel}$	roc_2	$\|\tilde{C} - C\|_{\infty,rel}$	roc_∞
$1 \cdot 10^3$	$1.32 \cdot 10^{-1}$		$1.21 \cdot 10^{-1}$	
$1 \cdot 10^4$	$3.72 \cdot 10^{-2}$	0.55	$3.95 \cdot 10^{-2}$	0.48
$1 \cdot 10^5$	$1.39 \cdot 10^{-2}$	0.43	$1.35 \cdot 10^{-2}$	0.47
$1 \cdot 10^6$	$4.45 \cdot 10^{-3}$	0.49	$5.69 \cdot 10^{-3}$	0.37

Table 2: Error (and rate of convergence, *roc*) on the covariance matrix $\mathbf{C}$ ($N = 72k$) *w.r.t.* the number of realizations n_{real} for a Gaussian correlation with $\ell = 0.25$ (top) and $\ell = 0.50$ (bottom). The approximate square root used to generate realizations is computed by a fixed accuracy randomized *SVD* ($\varepsilon = 10^{-2}$) powered by *ufmm* (left) or *smooth-ufmm* (right) with $p = 4$. The sample covariance $\tilde{\mathbf{C}}$ is computed from the n_{real} realizations as Eq. 12. Observations: The convergence rate is close to the theoretical rate, *i.e.*, 1/2. The rate slightly decreases as N grows, since C_r fails to represent C with sufficient accuracy, though the experimental covariance accuracy is already very satisfying, namely below 10%.

Acknowledgment

This work was supported by the Inria-Stanford associate team FastLA. Experiments presented in this paper were carried out using the PLAFRIM experimental testbed, being developed under the Inria PlaFRIM development action with support from LABRI and IMB and other entities: Conseil Régional d'Aquitaine, FeDER, Université de Bordeaux and CNRS (see https://plafrim.bordeaux.inria.fr/).

6 Fast matrix multiplication algorithms

This appendix gathers the fast hierarchical matrix multiplication algorithms used in the present paper.

6.1 The Black-Box FMM

Algorithm 2 corresponds to the original *bbfmm* algorithm as introduced by Fong and Darve [21].

Algorithm 2: *black-box FMM (bbfmm)*

Input: Kernel $k(\cdot,\cdot)$, Densities $\mathbf{w}$, Positions $\mathbf{x}$, interpolation order p, leaf level $\bar{L}$
Output: Potentials $\mathbf{f}$

```
// Precomputation
```
for *level* $L = 2, \dots, \bar{L}$ **do**
  `// Assemble M2L operators given a fictitious target cell` $\mathcal{C}_{\mathbf{x}}^{(L)}$
  for *source cell* $\mathcal{C}_{\mathbf{y}}^{(L)} \in \mathcal{I}(\mathcal{C}_{\mathbf{x}}^{(L)})$ **do**
    $K_{\boldsymbol{\alpha}\boldsymbol{\beta}} = k(\bar{\mathbf{x}}_{\boldsymbol{\alpha}}, \bar{\mathbf{y}}_{\boldsymbol{\beta}}),\ \forall \boldsymbol{\alpha}, \boldsymbol{\beta} / |\boldsymbol{\alpha}|, |\boldsymbol{\beta}| \leq p$

```
// Upward pass
```
for *level* $L = \bar{L}, \dots, 2$ **do**
  for *source cell* $\mathcal{C}_{\mathbf{y}}^{(L)} \in$ *tree* **do**
    `// P2M/M2M`
    if $L = \bar{L}$ **then**
      `// P2M: interpolation`
      $\mathcal{M}_{\boldsymbol{\beta}}^{(\bar{L})} = \sum_{j=1}^{N} S_{\boldsymbol{\beta}}^{p}(\mathbf{y}_j) w(\mathbf{y}_j),\ \forall \boldsymbol{\beta} / |\boldsymbol{\beta}| \leq p$
    else
      `// M2M:`
      $\mathcal{M}_{\boldsymbol{\beta}}^{(L)} = \sum_{|\boldsymbol{\beta'}| \leq p} S_{\boldsymbol{\beta}}^{p}(\bar{\mathbf{y}}_{\boldsymbol{\beta'}}) \mathcal{M}_{\boldsymbol{\beta'}}^{(L+1)},\ \forall \boldsymbol{\beta} / |\boldsymbol{\beta}| \leq p$

```
// Downward pass
```
for *level* $L = 2, \dots, \bar{L}$ **do**
  for *target cell* $\mathcal{C}_{\mathbf{x}}^{(L)} \in$ *tree* **do**
    `// M2L: transfer between interacting cells`
    for *source cell* $\mathcal{C}_{\mathbf{y}}^{(L)} \in \mathcal{I}(\mathcal{C}_{\mathbf{x}}^{(L)})$ **do**
      $\mathcal{L}_{\boldsymbol{\alpha}}^{(L)} += \sum_{|\boldsymbol{\beta}| \leq p} K_{\boldsymbol{\alpha}\boldsymbol{\beta}} \mathcal{M}_{\boldsymbol{\beta}}^{(L)},\ \forall \boldsymbol{\alpha} / |\boldsymbol{\alpha}| \leq p$
    `// L2L/L2P/P2P`
    if $L \leq \bar{L}$ **then**
      `// L2L:`
      $\mathcal{L}_{\boldsymbol{\alpha}}^{(L)} = \sum_{|\boldsymbol{\alpha'}| \leq p} S_{\boldsymbol{\alpha}}^{p}(\bar{\mathbf{x}}_{\boldsymbol{\alpha'}}) \mathcal{L}_{\boldsymbol{\alpha'}}^{(L-1)},\ \forall \boldsymbol{\alpha} / |\boldsymbol{\alpha}| \leq p$
    else
      `// L2P: interpolation`
      $f(\mathbf{x}_i) = \sum_{|\boldsymbol{\alpha}| \leq p} S_{\boldsymbol{\alpha}}^{p}(\mathbf{x}_i) \mathcal{L}_{\boldsymbol{\alpha}}^{(\bar{L})}$
      `// P2P: direct computation`
      for *source cell* $\mathcal{C}_{\mathbf{y}}^{(L)} \in \mathcal{N}(\mathcal{C}_{\mathbf{x}}^{(L)})$ **do**
        $f(\mathbf{x}_i) = \sum_{j=1}^{N} k(\mathbf{x}_i, \mathbf{y}_j) w(\mathbf{y}_j)$

6.2 The Uniform FMM

Algorithm 3 corresponds to the new matrix multiplication algorithm introduced in section 2.

Algorithm 3: *uniform FMM (ufmm)*

Input: Kernel $k(\cdot,\cdot)$, Densities $\mathbf{w}$, Positions $\mathbf{x}$, interpolation order p, leaf level $\bar{L}$
Output: Potentials $\mathbf{f}$

```
// Precomputation
```
for *level* $L = 2, \dots, \bar{L}$ **do**
  `// Assemble M2L operators (in Fourier domain)`
  for *source cell* $\mathcal{C}_{\mathbf{y}}^{(L)} \in \mathcal{I}(\mathcal{C}_{\mathbf{x}}^{(L)})$ **do**
    `// Compute first row of 3D block Toeplitz M2L operators`
    $R_{\boldsymbol{\beta}} = k(\bar{\mathbf{x}}_{\mathbf{0}}, \bar{\mathbf{y}}_{\boldsymbol{\beta}}),\ \forall \boldsymbol{\beta}/|\boldsymbol{\beta}| \leq p$
    `// Embed` R `in the first row` $\tilde{R}$ `of a 3D block circulant matrix`
    for $\boldsymbol{\beta}/|\boldsymbol{\beta}| < \tilde{p} = 2p$ **do**
      if $\beta_d > p$ **then**
        $\tilde{R}_{\boldsymbol{\beta}} = R_{\boldsymbol{\beta}'}$ with $\beta'_d = 2p - \beta_d$ and $\beta'_i = \beta_i$ for $i \neq d$
      else
        $\tilde{R}_{\boldsymbol{\beta}} = R_{\boldsymbol{\beta}}$
    `// Apply 3D DFT`
    $\hat{\tilde{\mathbf{R}}} = \mathbf{F}\tilde{\mathbf{R}}$ with $F_{\boldsymbol{\alpha}\boldsymbol{\beta}} = e^{-\frac{2i\pi}{\tilde{p}}(\boldsymbol{\alpha}\cdot\boldsymbol{\beta})},\ \forall(\boldsymbol{\alpha},\boldsymbol{\beta})/|\boldsymbol{\alpha}|, |\boldsymbol{\beta}| < \tilde{p}$
```
// Upward pass
```
for *level* $L = \bar{L}, \dots, 2$ **do**
  for *source cell* $\mathcal{C}_{\mathbf{x}}^{(L)} \in$ *tree* **do**
    `// P2M/M2M`
    `// ... see Algo. 2`
    `// Pad expansion with zeros and transfer to Fourier domain.`
    $\tilde{\boldsymbol{\mathcal{M}}}^{(L)} = \mathbf{0}_{\tilde{\mathbf{p}}}$
    $\tilde{\boldsymbol{\mathcal{M}}}_{\boldsymbol{\beta}}^{(L)} = \boldsymbol{\mathcal{M}}_{\boldsymbol{\beta}}^{(L)},\ \forall \boldsymbol{\beta}/|\boldsymbol{\beta}| \leq p$
    $\hat{\tilde{\boldsymbol{\mathcal{M}}}}^{(L)} = \mathbf{F}\tilde{\boldsymbol{\mathcal{M}}}^{(L)}$
```
// Downward pass
```
for *level* $L = 2, \dots, \bar{L}$ **do**
  for *target cell* $C_x \in$ *tree* **do**
    `// M2L: transfer between interacting cells`
    for *source cell* $\mathcal{C}_{\mathbf{y}}^{(L)} \in \mathcal{I}(\mathcal{C}_{\mathbf{x}}^{(L)})$ **do**
      $\hat{\tilde{\mathcal{L}}}_{\boldsymbol{\alpha}}^{(L)} += \hat{\tilde{R}}_{\boldsymbol{\alpha}} \hat{\tilde{\mathcal{M}}}_{\boldsymbol{\alpha}}^{(L)},\ \forall \boldsymbol{\alpha}/|\boldsymbol{\alpha}| < \tilde{p}$
    `// L2L/L2P/P2P`
    `// ... see Algo. 2`
    `// Transfer back to physical domain and unpad.`
    $\tilde{\boldsymbol{\mathcal{L}}}^{(L)} = \mathbf{F}^{-1}\hat{\tilde{\boldsymbol{\mathcal{L}}}}^{(L)}$
    $\boldsymbol{\mathcal{L}}_{\boldsymbol{\alpha}}^{(L)} = \tilde{\boldsymbol{\mathcal{L}}}_{\boldsymbol{\alpha}}^{(L)},\ \forall \boldsymbol{\alpha}/|\boldsymbol{\alpha}| \leq p$

7 Convergence of the hierarchical methods *w.r.t.* the point distribution

Here we present comparative results on the convergence of the *ufmm* and *smooth-ufmm* with respect to the interpolation order p for various geometries (unit sphere, cube, prolate sphere and hyperbolic paraboloid). In particular we analyze the influence of the length scale on the MVP error for the Gaussian correlation kernel. For all geometries the width of the bounding box is equal to 2. All computations where performed on a cluster computer, namely plafrim/mirabelle: Hexa-core Westmere Intel Xeon X5670 @ 2.93GHz with 96GB ram and 12MB L3 Cache.

Observations The *global fft* will always have the same cost at a given interpolation order, since this method is oblivious of the shape of the distribution. On the other hand, the cost of the hierarchical methods may vary significantly from one distribution to another. Let us for instance consider a Gaussian correlation with $\ell = 0.5$. If the particles are distributed in the cube (*i.e.*, an homogeneous distribution) the cost of the *global fft* lies somewhere between the *ufmm* and the *smooth-ufmm* with optimal h, see Figure 9. If the particles are distributed on a sphere (*i.e.*, an heterogeneous distribution) then the hierarchical methods become faster than the *global fft*, see Figure 10.

Other distributions Fig. 11 and Fig. 12 respectively confirm the previous observations on the prolate ellipsoid and the hyperbolic paraboloid, *i.e.*, highly heterogeneous distributions. The associated octrees have larger depths ($h = 7$) than for the unit sphere ($h = 5$), however in the lowest level a large number of cells are empty. Consequently, the computational times are only slightly larger than for the unit sphere.

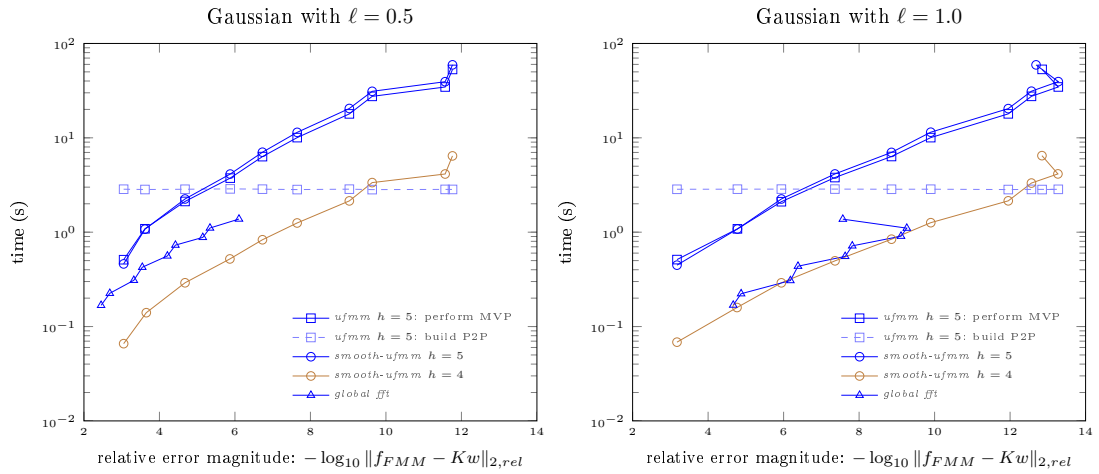


Figure 9: Computational time of a MVP *w.r.t.* the relative error magnitude using various algorithms: *ufmm* and *smooth-ufmm* with $p = 2 \dots 12$, and *global fft* with $p = 7 \dots 15$. We used $72k$ particles randomly distributed in a cube. Observations: *ufmm* and *smooth-ufmm* have approximately the same cost when they share the same tree depth. The *smooth-ufmm* is significantly faster if we choose an optimal tree depth, *e.g.*, $h = 4$ represented in brown. If the Gaussian decreases sufficiently slow (*i.e.*, the rank is sufficiently low), then the cost of the *global fft* is similar to the optimal *smooth-ufmm*. However, the *global fft* exhibits an instability for the highest interpolation order.

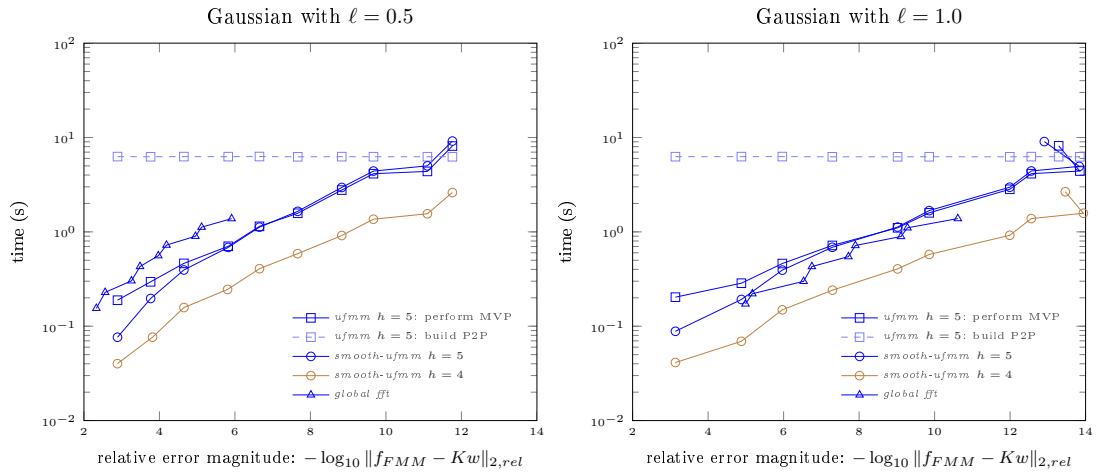


Figure 10: Computational time of a MVP *w.r.t.* the relative error magnitude using various algorithms: *ufmm* and *smooth-ufmm* with $p = 2 \dots 12$, and *global fft* with $p = 7 \dots 15$. We used $72k$ particles randomly distributed on the unit sphere. Observations: *ufmm* and *smooth-ufmm* have approximately the same cost when they share the same tree depth. The *smooth-ufmm* is significantly faster if we choose an optimal tree depth, *e.g.*, $h = 4$ represented in brown. If the Gaussian decreases sufficiently slow (*i.e.*, the rank is sufficiently low), then the cost of the *global fft* is slightly lower than the *ufmm* but the optimal *smooth-ufmm* still performs better.

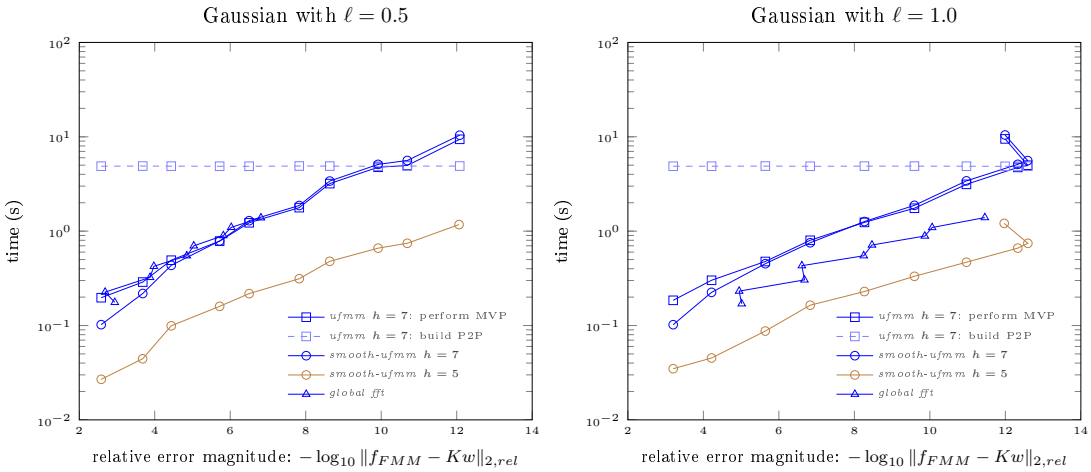


Figure 11: Computational time of a MVP *w.r.t.* the relative error magnitude using various algorithms: *ufmm* and *smooth-ufmm* with $p = 2 \dots 12$, and *global fft* with $p = 7 \dots 15$. We used $72k$ particles randomly distributed on a prolate ellipsoid (ratio 1:1:10). Observations: *ufmm* and *smooth-ufmm* have approximately the same cost when they share the same tree depth. The *smooth-ufmm* is significantly faster if we choose an optimal tree depth, *e.g.*, $h = 5$ represented in brown. If the Gaussian decreases sufficiently slow (*i.e.*, the rank is sufficiently low), then the *global fft* performs relatively well compared to the hierarchical variants.

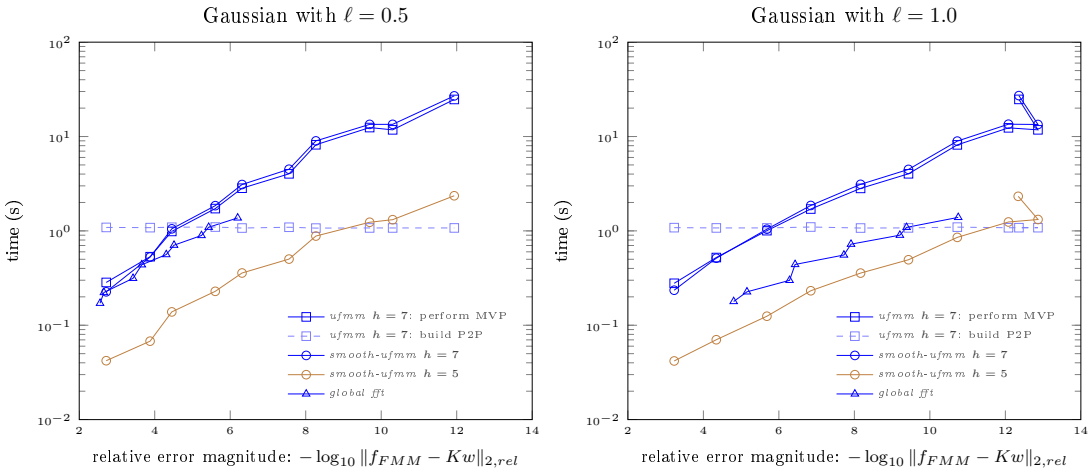


Figure 12: Computational time of a MVP *w.r.t.* the relative error magnitude using various algorithms: *ufmm* and *smooth-ufmm* with $p = 2 \dots 12$, and *global fft* with $p = 7 \dots 15$. We used $72k$ particles randomly distributed on a hyperbolic paraboloid (ratio 10:10:1). Observations: *ufmm* and *smooth-ufmm* have approximately the same cost when they share the same tree depth. The *smooth-ufmm* is significantly faster if we choose an optimal tree depth, *e.g.*, $h = 5$ represented in brown. If the Gaussian decreases sufficiently slow (*i.e.*, the rank is sufficiently low), then the *global fft* performs relatively well compared to the hierarchical variants.

References

[1] Dimitris Achlioptas, *Database-friendly random projections: Johnson-lindenstrauss with binary coins*, Journal of computer and System Sciences, 66 (2003), pp. 671–687.

[2] Emmanuel Agullo, Bérenger Bramas, Olivier Coulaud, Eric Darve, Matthias Messner, and Toru Takahashi, *Pipelining the fast multipole method over a runtime system*, arXiv preprint arXiv:1206.0115, (2012).

[3] ——, *Task-based fmm for multicore architectures*, SIAM Journal on Scientific Computing, 36 (2014), pp. C66–C93.

[4] Nir Ailon and Bernard Chazelle, *The fast johnson-lindenstrauss transform and approximate nearest neighbors*, SIAM Journal on Computing, 39 (2009), pp. 302–322.

[5] Adolfo J Banchio and John F Brady, *Accelerated stokesian dynamics: Brownian motion*, The Journal of chemical physics, 118 (2003), pp. 10323–10332.

[6] Alberto Bellin and Yoram Rubin, *Hydro_gen: A spatially distributed random field generator for correlated properties*, Stochastic Hydrology and Hydraulics, 10 (1996), pp. 253–278.

[7] Edmund Bertschinger, *Multiscale gaussian random fields and their application to cosmological simulations*, The astrophysical journal supplement series, 137 (2001), p. 1.

[8] Pierre Blanchard, Bérenger Bramas, Olivier Coulaud, Eric Darve, Laurent Dupuy, Arnaud Etcheverry, and Guillaume Sylvand, *Scalfmm: A generic parallel fast multipole library*, in Computational Science and Engineering (CSE), SIAM, Mar. 2015.

[9] John P Boyd, *Defeating the runge phenomenon for equispaced polynomial interpolation via tikhonov regularization*, Applied Mathematics Letters, 5 (1992), pp. 57–59.

[10] ——, *Trouble with gegenbauer reconstruction for defeating gibbs' phenomenon: Runge phenomenon in the diagonal limit of gegenbauer polynomial approximations*, Journal of Computational Physics, 204 (2005), pp. 253–264.

[11] John P Boyd and Jun Rong Ong, *Exponentially-convergent strategies for defeating the runge phenomenon for the approximation of non-periodic functions, part i: Single-interval schemes. commun*, Comput. Phys, 5 (2009), pp. 484–497.

[12] ——, *Exponentially-convergent strategies for defeating the runge phenomenon for the approximation of non-periodic functions, part two: Multi-interval polynomial schemes and multidomain chebyshev interpolation*, Applied Numerical Mathematics, 61 (2011), pp. 460–472.

[13] John P Boyd and Fei Xu, *Divergence (runge phenomenon) for least-squares polynomial approximation on an equispaced grid and mock–chebyshev subset interpolation*, Applied Mathematics and Computation, 210 (2009), pp. 158–168.

[14] Julien Carron, Melody Wolk, and Istvan Szapudi, *On fast generation of cosmological random fields*, Monthly Notices of the Royal Astronomical Society, 444 (2014), pp. 994–1000.

[15] Michael W Davis, *Generating large stochastic simulations—the matrix polynomial approximation method*, Mathematical Geology, 19 (1987), pp. 99–107.

[16] ——, *Production of conditional simulations via the lu triangular decomposition of the covariance matrix*, Mathematical Geology, 19 (1987), pp. 91–98.

[17] Vahid Dehdari and Clayton V. Deutsch, *Applications of randomized methods for decomposing and simulating from large covariance matrices*, in Geostatistics Oslo 2012, Petter Abrahamsen, Ragnar Hauge, and Odd Kolbjørnsen, eds., vol. 17 of Quantitative Geology and Geostatistics, Springer Netherlands, 2012, pp. 15–26.

[18] CR Dietrich and GN Newsam, *A fast and exact method for multidimensional gaussian stochastic simulations*, Water Resources Research, 29 (1993), pp. 2861–2869.

[19] Petros Drineas and Michael W Mahoney, *On the nyström method for approximating a gram matrix for improved kernel-based learning*, The Journal of Machine Learning Research, 6 (2005), pp. 2153–2175.

[20] William D Elliott and John A Board, Jr, *Fast fourier transform accelerated fast multipole algorithm*, SIAM Journal on Scientific Computing, 17 (1996), pp. 398–415.

[21] William Fong and Eric Darve, *The black-box fast multipole method*, Journal of Computational Physics, 228 (2009), pp. 8712–8725.

[22] Alex Gittens and Michael W Mahoney, *Revisiting the nystrom method for improved large-scale machine learning*, arXiv preprint arXiv:1303.1849, (2013).

[23] Leslie Greengard and Vladimir Rokhlin, *A fast algorithm for particle simulations*, Journal of computational physics, 73 (1987), pp. 325–348.

[24] ——, *On the efficient implementation of the fast multipole algorithm*, Yale University, Department of Computer Science, 1988.

[25] A Gutjahr, D McKay, and JL Wilson, *Fast fourier transform methods for random field generation*, Eos Trans. AGU, 68 (1987), p. 1265.

[26] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM review, 53 (2011), pp. 217–288.

[27] Mark E Johnson, *Multivariate statistical simulation: A guide to selecting and generating continuous multivariate distributions*, John Wiley & Sons, 2013.

[28] Andre G Journel, *Geostatistics for conditional simulation of ore bodies*, Economic Geology, 69 (1974), pp. 673–687.

[29] PK Kitanidis and J Lee, *Principal component geostatistical approach for large-dimensional inverse problems*, Water resources research, 50 (2014), pp. 5428–5443.

[30] Mickaële Le Ravalec, Benoît Noetinger, and Lin Y Hu, *The fft moving average (fft-ma) generator: An efficient numerical method for generating and conditioning gaussian simulations*, Mathematical Geology, 32 (2000), pp. 701–723.

[31] Edo Liberty, Nir Ailon, and Amit Singer, *Dense fast random projections and lean walsh transforms*, in Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques, Springer, 2008, pp. 512–522.

[32] Edo Liberty, Franco Woolfe, Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert, *Randomized algorithms for the low-rank approximation of matrices*, Proceedings of the National Academy of Sciences, 104 (2007), pp. 20167–20172.

[33] Michael W Mahoney, *Randomized algorithms for matrices and data*, Foundations and Trends® in Machine Learning, 3 (2011), pp. 123–224.

[34] Michael W Mahoney and Petros Drineas, *Cur matrix decompositions for improved data analysis*, Proceedings of the National Academy of Sciences, 106 (2009), pp. 697–702.

[35] Aristotelis Mantoglou, *Digital simulation of multivariate two-and three-dimensional stochastic processes with a spectral turning bands method*, Mathematical Geology, 19 (1987), pp. 129–149.

[36] Aristotelis Mantoglou and John L Wilson, *The turning bands method for simulation of random fields using line generation by a spectral method*, Water Resources Research, 18 (1982), pp. 1379–1394.

[37] Per-Gunnar Martinsson, *A fast randomized algorithm for computing a hierarchically semiseparable representation of a matrix*, SIAM Journal on Matrix Analysis and Applications, 32 (2011), pp. 1251–1274.

[38] Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert, *A randomized algorithm for the approximation of matrices*, tech. report, DTIC Document, 2006.

[39] Matthias Messner, Bérenger Bramas, Olivier Coulaud, and Eric Darve, *Optimized m2l kernels for the chebyshev interpolation based fast multipole method*, arXiv preprint arXiv:1210.7292, (2012).

[40] Dean S Oliver, *Moving averages for gaussian simulation in two and three dimensions*, Mathematical Geology, 27 (1995), pp. 939–960.

[41] V. Y. Pan, J. H. Reif, and S. R. Tate, *The power of combining the techniques of algebraic and numerical computing: Improved approximate multipoint polynomial evaluation and improved multipole algorithms*, in Proceedings of the 33rd Annual Symposium on Foundations of Computer Science, SFCS '92, Washington, DC, USA, 1992, IEEE Computer Society, pp. 703–713.

[42] Ue-Li Pen, *Generating cosmological gaussian random fields*, The Astrophysical Journal Letters, 490 (1997), p. L127.

[43] MJL Robin, AL Gutjahr, EA Sudicky, and JL Wilson, *Cross-correlated random field generation with the direct fourier transform method*, Water Resources Research, 29 (1993), pp. 2385–2397.

[44] Tamas Sarlos, *Improved approximation algorithms for large matrices via random projections*, in 47th Annual IEEE Symposium on Foundations of Computer Science, 2006, FOCS'06., IEEE, 2006, pp. 143–152.

[45] Masanobu Shinozuka and C-M Jan, *Digital simulation of random processes and its applications*, Journal of sound and vibration, 25 (1972), pp. 111–128.

[46] Si Si, Cho-Jui Hsieh, and Inderjit Dhillon, *Memory efficient kernel approximation*, in Proceedings of The 31st International Conference on Machine Learning, 2014, pp. 701–709.

[47] Joel A Tropp, *Improved analysis of the subsampled randomized hadamard transform*, Advances in Adaptive Data Analysis, 3 (2011), pp. 115–126.

[48] Peng Wang, Chunhua Shen, and Anton van den Hengel, *Efficient sdp inference for fully-connected crfs based on low-rank decomposition*, arXiv preprint arXiv:1504.01492, (2015).

[49] Ruoxi Wang, Yingzhou Li, Michael W Mahoney, and Eric Darve, *Structured block basis factorization for scalable kernel matrix evaluation*, arXiv preprint arXiv:1505.00398, (2015).

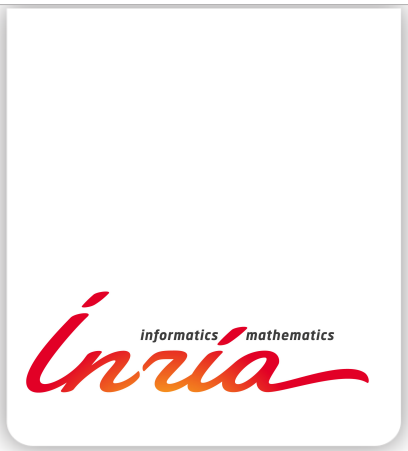
[50] Christopher Williams and Matthias Seeger, *Using the nyström method to speed up kernel machines*, in Proceedings of the 14th Annual Conference on Neural Information Processing Systems, no. EPFL-CONF-161322, 2001, pp. 682–688.

[51] Andrew TA Wood and Grace Chan, *Simulation of stationary gaussian processes in [0, 1] d*, Journal of computational and graphical statistics, 3 (1994), pp. 409–432.

[52] David P Woodruff, *Sketching as a tool for numerical linear algebra*, arXiv preprint arXiv:1411.4357, (2014).

[53] Franco Woolfe, Edo Liberty, Vladimir Rokhlin, and Mark Tygert, *A fast randomized algorithm for the approximation of matrices*, Applied and Computational Harmonic Analysis, 25 (2008), pp. 335 – 366.

[54] Kai Zhang, Ivor W Tsang, and James T Kwok, *Improved nyström low-rank approximation and error analysis*, in Proceedings of the 25th international conference on Machine learning, ACM, 2008, pp. 1232–1239.



RESEARCH CENTRE
BORDEAUX – SUD-OUEST

200 avenue de la Vieille Tour
33405 Talence Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399